

Automated Bidding in Computing Service Markets
Strategies, Architectures, Protocols

Zur Erlangung des akademischen Grades eines
Doktors der Wirtschaftswissenschaften

(Dr. rer. pol.)

von der Fakultät für Wirtschaftswissenschaften
des Karlsruher Institut für Technologie (KIT)

genehmigte

DISSERTATION

von

Dipl.-Inform. Nikolay Nikolaev Borissov

Tag der mündlichen Prüfung: 25. November 2011

Referent: Prof. Dr. Christof Weinhardt

Korreferent: Prof. Dr. Dirk Neumann

Karlsruhe 2012

Abstract

Interdisciplinary research in computer science and economics shows that great advancements are taking place in the areas of distributed computing, social networks, market mechanisms, game theory and artificial intelligence. Prominent examples of these developments are search engines like Google search, knowledge engines like WolframAlpha, electronic markets like eBay, Cloud computing markets like Amazon's EC2 Spot Instances, Google's AppEngine and SpotCloud, social networks like Facebook and LinkedIn, and many more.

Market mechanisms are studied for years to determine how goods and services can be allocated to consumers efficiently. In economic theory, market mechanisms often refer to design incentives and pricing to achieve a common goal by guiding the behavior of self-interested agents. In the field of market-based scheduling, consumers often demand and use different applications, providers offer heterogeneous computing services with different business models, therefore, the behavior and goals of both parties on the market can be highly heterogeneous. The research on bidding strategies is prevalently based on simplified assumptions about the type of agents, which are often assumed to be homogeneous, rational and sharing their true preferences with each other. Novel interdisciplinary research in Computational Mechanism Design aims to relax these assumptions by enhancing promising economic theories and evaluating them extensively in more realistic experiments by utilizing a high number of computing services over the Internet.

This work contributes to the research on Computational Mechanism Design by providing novel theoretical and software models – a novel bidding strategy called Q-Strategy, which automates bidding processes in imperfect information markets, a software framework for realizing agents and bidding strategies called BidGenerator and a communication protocol called MX/CS, for expressing and exchanging economic and technical information in a market-based scheduling system. The interdisciplinary approach to this research deals with the areas of design science, game-theoretic modeling, mechanism design, software engineering, agent-based modeling, discrete event simulation, scientific computing on cluster machines and statistical analysis. This work provides a full-fledged analysis of bidding strategies, bidding agents and communication protocols and is based on commonly applied methodologies for agent-based and technical evaluations. Evaluation of the Q-Strategy against benchmark strategies

in spot market scenarios showed that, on average, agents applying the Q-Strategy outperformed benchmark strategies in homogeneous and heterogeneous competition settings. As proof-of-concept, BidGenerator, the Q-Strategy and MX/CS have been implemented and integrated in a real running prototype for market-based scheduling and three real application case studies, which are all part of the SORMA project.

To my love, Petya, my brother, Borislav, and my parents, Ana and Nikolay.

Acknowledgments

I am deeply grateful for the support and contributions of many people without whom this dissertation would not have been possible.

Foremost, I would like to express my sincere gratitude to my supervisor, Professor Dr. Christof Weinhardt, for his continuous support and patience, and for the freedom that allowed me to pursue my PhD research. I was very pleased to be a part of his Information and Market Engineering group at the Institute of Information Systems and Management (IISM) and for the opportunity he gave me to participate in the European research project SORMA. I would like to thank Professor Dr. Dirk Neumann for co-advising this thesis and guiding me during the starting phase of my research. I am very grateful for his allowing me the opportunity to work with great people, and to grow and learn with various technical and administrative responsibilities within the SORMA project. I appreciate the support of my dissertation committee and the great atmosphere provided by the examiners, Professor Dr. Andreas Oberweis and Professor Dr. Martin Ruckes.

I owe my deepest gratitude to Dr. Simon Caton who accompanied me during the final phase of my PhD research by proofreading major parts of my work and for offering critical and constructive questions and comments. Special thanks go to Dr. Wibke Michalk, Dr. Thomas Meinel, Dr. Arun Anandasivam, and Dr. Carsten Block for proofreading parts of this work.

I am indebted to many of my colleagues in the Information & Market Engineering group at the Institute of Information Systems and Management (IISM), Forschungszentrum Informatik (FZI) and Karlsruhe Service Research Institute (KSRI) for their continuous support, feedback, and valuable discussions. It was also an honor for me to work with the colleagues from the SORMA project. I will always remember with fondness the times when we met in Barcelona to code and prepare the project's prototype and deliverables. I would especially like to thank Dr. Garry Smith, Professor Dr. Jens Nimis, Dr. Simon Caton, Niklas Wirström, Dr. René Brunner, Mario Macías, Pablo Chacin, and the rest of the development team. It was a pleasure code, publish, learn from and with you.

Thanks to Christian Haas and all contributors for the nice graduation cap. I love it.

I am very indebted to my family, who provided me with the tools I have needed to succeed in life, along with the support and courage I have needed to overcome life's challenges. My greatest thanks go to my wife, Petya, for giving me her love and patience. Her unwavering support has allowed me to successfully complete this research.

Nikolay Borissov

Table of Contents

Abstract	iii
Acknowledgments	vii
List of Tables	xvi
List of Figures	xviii
List of Notations	xxi
Part I Foundations	1
Chapter 1. Introduction	3
1.1 Motivation	4
1.2 Research Outline	6
1.3 Summary of the Research Contributions	12
1.4 Structure of This Work	14
1.5 Related Publications	16
Chapter 2. Preliminaries and Related Work	17
2.1 Computing Services	17
2.1.1 Utility Computing	18
2.1.2 Grid Computing	19
2.1.3 Cloud Computing	22
2.1.3.1 Infrastructure as a Service	24
2.1.3.2 Deployment Models	27
2.1.4 Concluding Remarks	30
2.2 Bidding, Agents and Messages	31
2.3 The Microeconomic Environment	36
2.3.1 The Microeconomic System	36

2.3.2	Classification of the Transaction Object	41
2.3.3	Economic Challenges	43
2.3.4	Market Mechanisms for Scheduling Computing Services	45
2.3.5	The Continuous Double Auction	48
2.3.6	Consumer and Provider Bidding Strategies	49
2.4	The Technical Environment	51
2.4.1	Application Scenarios	52
2.4.1.1	Batch Supply Chain Data Analysis	52
2.4.1.2	Interactive Sensor Data Analysis	54
2.4.2	Technical Challenges for the Market System	56
2.4.3	A Technical Architecture for Market-Based Scheduling	59
2.4.3.1	Layer 1: Core Market Services	60
2.4.3.2	Layer 2: Open Market Middleware	61
2.4.3.3	Layer 3: Consumer and Provider Bidding Tools	62
2.4.3.4	Layer 4: Applications and Resources	63
2.4.4	Bidding Scenario for Computing Services	63
2.5	Research Methods	65

Part II Design and Implementation 69

Chapter 3.	Economic Design of Bidding Strategies	71
3.1	What is a Bidding Strategy?	72
3.2	Design Desiderata	74
3.3	Existing Bidding Strategies	77
3.3.1	Bidding in Games with Perfect Information	77
3.3.2	Bidding in Games with Imperfect Information	78
3.3.3	Non-Adaptive Bidding Strategies	81
3.3.4	Adaptive Bidding Strategies	82
3.3.5	Analytical Comparison to the Desiderata	90
3.4	The Q-Strategy	97
3.4.1	Why Reinforcement Learning-Based Bidding?	97
3.4.2	The General Q-Learning Framework	98
3.4.3	The Q-Strategy Model	101
3.4.3.1	Automating the Bidding Processes	101

3.4.3.2	State Representation	101
3.4.3.3	Action Representation and Adaptive Bidding	103
3.4.3.4	Goal Representation and Q-Value Update	105
3.4.3.5	Realization of the Q-Strategy	106
3.4.3.6	Complexity Analysis	108
3.4.3.7	Critical View	109
3.5	Summary	110
Chapter 4.	Architectural Design of a Bidding Framework	112
4.1	What is a Bidding Agent?	112
4.2	Design Desiderata	114
4.3	Existing Agent Frameworks	117
4.3.1	Selection of Existing Agent Frameworks	117
4.3.2	Analytical Comparison to the Desiderata	119
4.4	Architecture of the Bidding Agent Framework	125
4.4.1	Decoupling of Market Platform and Bidding Agents	125
4.4.2	Methodic Realization of Agents and Bidding Strategies	127
4.4.3	Using Well-Defined Communication Protocols	131
4.4.4	Applicable in a Market-based Scheduling Domain	132
4.4.5	Integrating Market Information Service	133
4.4.6	Supporting Secured and Trusted Communication	134
4.5	Summary	136
Chapter 5.	Communication Protocols	138
5.1	Design Desiderata	139
5.2	Existing Communication Protocols	142
5.2.1	Selection of the Communication Protocols	142
5.2.2	Analytical Comparison to the Desiderata	144
5.3	Message Exchange in Computing Service Markets	146
5.3.1	Private Message	148
5.3.2	State Message	150
5.3.3	Public Message	151
5.3.4	Market Message	151
5.3.5	Market Information Message	153
5.4	Matchmaking of Public Messages	154

5.5	The MX/CS Ontology	157
5.6	Extensions for Combinatorial Bids	159
5.7	Summary	160
Part III Evaluation		163
Chapter 6. Agent-Based Numerical Experiments		165
6.1	Evaluation Methodology	165
6.1.1	Setup of Market and Application Data	166
6.1.2	Setup of Bidding Strategies	169
6.1.3	Setup of the Multi-Agent System	171
6.1.4	Analyzing the Efficiency of Bidding Strategies	173
6.1.5	Sensitivity Analysis	174
6.1.6	Correlation Analysis	174
6.2	Evaluation Results	176
6.2.1	Consumer Outcomes	176
6.2.2	Provider Outcomes	187
6.2.3	Combined Outcomes	193
6.3	Summary	200
Chapter 7. Technical Analysis and Application		201
7.1	Technical Analysis as Methodology	201
7.2	Case Study 1: Batch Supply Chain Data Analysis	202
7.2.1	TXTDemand and Its Market Scenario	202
7.2.2	TXTDemand Requirements	204
7.2.3	Integration of BidGenerator and <i>MX/CS</i> Protocol	205
7.2.4	Summary of Case Study Contributions	207
7.3	Case Study 2: Interactive Sensor Data Analysis	208
7.3.1	Visage and Its Market Scenario	208
7.3.2	Visage Requirements	209
7.3.3	Integration of the BidGenerator and <i>MX/CS</i> Protocol	209
7.3.4	Summary of Case Study Contributions	212
7.4	Case Study 3: e-Service Level Agreements	213
7.4.1	Service Level Agreements in SORMA and Requirements	213

7.4.2	Application of the <i>MX/CS</i> Protocol	214
7.4.3	Summary of Case Study Contributions	215
7.5	Further Application Scenarios of <i>MX/CS</i>	216
7.6	Performance Analysis	216
7.6.1	Monitoring of Distributed Services	216
7.6.2	Technical Performance Experiments	219
7.7	Summary	221
Part IV Conclusion		223
Chapter 8. Summary of This Work and Future Research		225
8.1	Summary of Contributions	225
8.2	Future Research	229
8.2.1	Hierarchical Bidding and Transfer Learning	229
8.2.2	Automated Bidding for Complex Service Mashups	230
8.2.3	Design of Flexible Market Platforms	231
8.2.4	Legal Issues and Matchmaking of Service Level Agreements	231
8.2.5	Information Services for Computing Service Markets	231
8.2.6	Economic Resource Management	232
8.2.7	Cloud Application Engineering and Standardization	232
8.2.8	Complementary Research	232
Appendices		234
Appendix A. Sensitivity Analysis		235
Appendix B. Full Tables of the Evaluation Results		240
B.1	Consumer Outcomes	240
B.2	Provider Outcomes	248
B.3	Combined Outcomes	256
Appendix C. <i>MX/CS</i> – Communication Protocol Specification		264
C.1	<i>MX/CS</i> – XML Schema Specification	264
C.2	<i>MX/CS</i> – OWL Specification	268
C.3	State Message and Market Information	282

List of Tables

2.1	Cloud deployment models	28
2.2	Computational mechanism design desiderata (Myerson and Satterthwaite, 1983; Krishna and Perry, 1998; Shneidman et al., 2005; Parkes, 2008)	44
3.1	Mapping of the desiderata for strategy design to existing bidding strategies	92
4.1	Standardization of agent frameworks	118
4.2	Comparison of the agent frameworks	120
5.1	Comparison of communication protocols	142
6.1	Design of the experiments and variables	166
6.2	Winning performance of the <i>Q-Strategy</i> against the benchmark strategies, ZIP and GD, in homogeneous and heterogeneous settings of consumers and providers	177
6.3	Top 10 consumer outcomes of settings with 50 providers and the LLNL data profile. The higher the combined CAAS, the better the outcome of the setting.	178
6.4	Top 10 consumer outcomes of settings with 100 providers and the LLNL data profile. The higher the combined CAAS, the better the outcome of the setting.	179
6.5	Top 10 consumer outcomes of settings with 50 providers and the HPC2N data profile. The higher the combined CAAS, the better the outcome of the setting.	179
6.6	Top 10 consumer outcomes of settings with 100 providers and the HPC2N data profile. The higher the combined CAAS, the better the outcome of the setting.	180
6.7	Top 10 provider outcomes of settings with 50 providers and the LLNL data profile. The higher the PAAS, the better the outcome of the setting.	188
6.8	Top 10 provider outcomes of settings with 100 providers and the LLNL data profile. The higher the PAAS, the better the outcome of the setting.	188
6.9	Top 10 provider outcomes of settings with 50 providers and the HPC2N data profile. The higher the PAAS, the better the outcome of the setting.	189

6.10	Top 10 provider outcomes of settings with 100 providers and the HPC2N data profile. The higher the PAAS, the better the outcome of the setting.	189
7.1	Detailed time performance analysis of the integrated system in settings with 20 and 40 Truth-Telling agents, as well as Q-Strategy agents	221
A.1	Selection of Q-Strategy's parameters in settings with Q-Strategy consumers and providers with the LLNL data profile.	236
A.2	Selection of ZIP-Strategy's parameters in settings with ZIP consumers and providers with the LLNL data profile.	237
A.3	Selection of GD-Strategy's parameters in settings with GD-Strategy consumers and providers with the LLNL data profile.	237
A.4	Selection of Q-Strategy's parameters in settings with Q-Strategy consumers and providers with the HPC2N data profile.	238
A.5	Selection of ZIP-Strategy's parameters in settings with ZIP consumers and providers with the HPC2N data profile.	239
A.6	Selection of GD-Strategy's parameters in settings with GD consumers and providers with the HPC2N data profile.	239
B.1	Consumer outcomes of settings with 50 providers and the LLNL workload.	240
B.2	Consumer outcomes of settings with 100 providers and the LLNL workload.	242
B.3	Consumer outcomes of settings with 50 providers and the HPC2N workload.	244
B.4	Consumer outcomes of settings with 100 providers and the HPC2N workload.	246
B.5	Provider outcomes of settings with 50 providers and the LLNL workload.	248
B.6	Provider outcomes of settings with 100 providers and the LLNL workload.	250
B.7	Provider outcomes of settings with 50 providers and the HPC2N workload.	252
B.8	Provider outcomes of settings with 100 providers and the HPC2N workload.	254
B.9	Combined outcomes of settings with 50 providers and the LLNL workload.	256
B.10	Combined outcomes of settings with 100 providers and the LLNL workload.	258
B.11	Combined outcomes of settings with 50 providers and the LLNL workload.	260
B.12	Combined outcomes of settings with 50 providers and the LLNL workload.	262

List of Figures

1.1	Structure of this work	15
2.1	Cloud services (own representation)	23
2.2	Static versus dynamic pricing (adopted from Lai (2005))	32
2.3	Consumers' (CS) and providers' surpluses (PS) out of equilibrium (adopted from Gjerstad (2007))	34
2.4	Microeconomic system (Neumann, 2007)	38
2.5	Scenario for supply chain data analysis with local and externally pur- chased computing services (own representation, based on the SORMA project) (Windsor et al., 2009; Nimis et al., 2008)	53
2.6	Scenario for on-demand video data analysis with local and externally purchased computing services (own representation, based on the SORMA project) (Windsor et al., 2009; Nimis et al., 2008)	55
2.7	SORMA architecture	60
2.8	Scenario for bidding on computing services	64
3.1	Standard reinforcement learning interaction loop (adopted from Kael- bling et al. (1996))	99
3.2	Example of Q-Strategy's multi-states with multi-action arms	104
3.3	Q-Strategy's implementation in BidGenerator	107
4.1	A bird's-eye view of the interactions between BidGenerator and related components	126
4.2	BidGenerator architecture	129
4.3	BidGenerator's dynamic view of bidding processes	130
4.4	BidGenerator communication components	132
4.5	BidGenerator security integration	135
5.1	Communication protocols for the automated provisioning and usage of computing services	147
6.1	Distribution of the job durations and cumulative distribution function (CDF) of the data profiles	168

6.2	Architecture of an agent-based experimental environment with a Discrete Event Service Engine	172
6.3	Consumer outcomes in settings with GD-Providers and the LLNL data profile.	181
6.4	Consumer outcomes in settings with GD-Providers and the HPC2N data profile.	182
6.5	Consumer outcomes in settings with ZIP-Providers and the LLNL data profile.	183
6.6	Consumer outcomes in settings with ZIP-Providers and the HPC2N data profile.	184
6.7	Consumer outcomes in settings with Q-Providers and the LLNL data profile.	185
6.8	Consumer outcomes in settings with Q-Providers and the HPC2N data profile.	186
6.9	Provider outcomes in settings with competing consumers and the LLNL data profile.	191
6.10	Provider outcomes in settings with competing consumers and the HPC2N data profile.	192
6.11	Combined consumer and provider outcomes in settings with GD-Providers and the LLNL data profile.	194
6.12	Combined consumer and provider outcomes in settings with GD-Providers and the HPC2N data profile.	195
6.13	Combined consumer and provider outcomes in settings with ZIP-Providers and the LLNL data profile.	196
6.14	Combined consumer and provider outcomes in settings with ZIP-Providers and the HPC2N data profile.	197
6.15	Combined consumer and provider outcomes in settings with Q-Providers and the LLNL data profile.	198
6.16	Combined consumer and provider outcomes in settings with Q-Providers and the HPC2N data profile.	199
7.1	TXTDemand: An application for supply chain management and data analysis (screenshot provided by the SORMA project)	202
7.2	Integrated view of TXTDemand SaaS, TXTOrchestrator, BidGenerator, MX/CS and the SORMA Market for Computing Services (own representation)	203
7.3	TXTOrchestrator: General view (screenshot provided by the SORMA project)	205
7.4	TXTOrchestrator: Job details (screenshot provided by the SORMA project)	206

7.5	Integrated view of TXTDemand SaaS, TXTOrchestrator, BidGenerator, MX/CS and the SORMA Market for Computing Services (own representation)	208
7.6	Visage client invoked BidGenerator with the Q-Strategy, received an allocation of Visage with an endpoint reference and started an analysis of video data sequences (screenshot provided by the SORMA project)	210
7.7	Visage integration (Garry Smith, demonstration at IES 2009, project SORMA)	211
7.8	Transformation of the MarketMessage concepts into a WS-Agreement (Borissov et al., 2009b).	215
7.9	Performance Monitor: System specific data with Java's VisualVM (own screenshot)	217
7.10	Distributed service process monitoring: SORMA Dashboard (screenshot provided by the SORMA project)	218
7.11	Time performance chart of the integrated system in settings with 20 and 40 Truth-Telling agents, as well as Q-Strategy agents	220

List of Notations

A, a	Action	79
B, b	Attribute	37
\mathcal{B}	Public message type	64
c	Duration	166
\mathcal{E}	Environment	36
E, e	Expected utility	80
F	Time to complete	166
G, g	Game	79
H, h	History	79
M, m	Bidding language	79
μ	Margin	82
N, n	Agent	79
Π	Payment method	149
Δ	Penalty	149
π	Price	152
\mathcal{P}	Private message type	148
R, r	Reward	99
ρ	Transition probability	79
Q, q	Bid	80
Ω, ω	State	102
\mathcal{S}	State message	150
S, s	Strategy	79
Υ	Signature	151
Θ	Technical description attributes	102
T, t	Time	79
θ	Transaction object	37
U, u	Score	105
V, v	Valuation	103
\mathcal{X}	Market message type	152
z	Expiration time	148

Part I

Foundations

Chapter 1

Introduction

INTERDISCIPLINARY research in computer science and economics shows that great advancements are taking place in the areas of distributed computing, social networks, market mechanisms, game theory and artificial intelligence (Blume, 2010). Prominent examples of these developments are search engines like Google search, knowledge engines like WolframAlpha, electronic markets like eBay, Cloud computing markets like Amazon’s EC2 Spot Instances, Google’s AppEngine and SpotCloud, social networks like Facebook and LinkedIn, and many more. Market mechanisms are studied for years to determine how goods and services can be allocated to consumers efficiently (Sutherland, 1968; Buyya, 2002; Heydenreich et al., 2010). In economic theory, market mechanisms often refer to design incentives and pricing to achieve a common goal by guiding the behavior of self-interested agents. In the field of market-based scheduling, consumers often demand and use different applications, providers offer heterogeneous computing services with different business models, therefore, the behavior and goals of both parties on the market can be highly heterogeneous (Blume, 2010; Levine, 2010). Moreover, bidding strategies implement individual decisions in software agents and are still based on simplified assumptions about the type of agent, which is often assumed to be homogeneous, as well as about the type of information available and it is often assumed that agents share their actions and allocations (Das et al., 2001; Reeves et al., 2005; Vytelingum et al., 2008; Schwartzman and Wellman, 2009). Novel interdisciplinary research in *Computational Mechanism Design* aims to relax these assumptions by enhancing promising economic theories and evaluating them extensively in more realistic experiments by utilizing a high number of computing services over the Internet (Parkes, 2008; EGEE, 2009).

This work contributes to the research on *Computational Mechanism Design* by providing novel theoretical and software models – a novel bidding strategy called *Q-Strategy*, which automates bidding processes in imperfect information markets, a software framework for realizing agents and bidding strategies called *BidGenerator* and a communication protocol called *MX/CS* for expressing and exchanging economic and technical information in a market-based scheduling system. The interdisciplinary

approach to this research deals with the areas of design science, game-theoretic modeling, mechanism design, software engineering, agent-based modeling, discrete event simulation, scientific computing on cluster machines and statistical analysis. This work provides a full-fledged analysis of bidding strategies, bidding agents and communication protocols and is based on commonly applied methodologies for agent-based and technical evaluations. Evaluation of the *Q-Strategy* against benchmark strategies in spot market scenarios showed that, on average, agents that apply *Q-Strategy* outperformed benchmark strategies in homogeneous and heterogeneous competition settings. As proof-of-concept, *BidGenerator*, the *Q-Strategy* and *MX/CS* have been implemented and integrated in a real running prototype for market-based scheduling and three real application case studies, which are all part of the SORMA project.

1.1 Motivation

The economics of computing service provisioning refers to mechanisms that improve the efficiencies of their realization and utilization. Studies have shown that, on average, 5% to 20% of computing center resources are utilized (Armbrust et al., 2009; Greenberg et al., 2008) and in companies, around 60% (Symantec, 2008). Current workload logs of the *MonALISA*¹ repositories confirmed these low average rates of utilization. The current top ten computing centers of the Top500 list² provide more than 12 petaFLOPS³ of computing power, which is used by research institutions and can also be made available to industry. Such a shift in provisioning to business has considerable implications for scheduling policies. Current cluster or Grid computing (decentralized coordination of connected cluster systems among distributed sites based on common interfaces and tools) resource managers execute technical and agreement-based policies rather than economic ones. Consumers of such resources are often grouped into virtual organizations with specific privileges and quotas for resource usage (Feitelson et al., 2005; Elmroth and Gardfjall, 2005; Elmroth et al., 2008). For example, the execution of a job depends on several technical factors, including the quotas and permissions granted to the consumer, the part of a virtual organization it is assigned to, the type and amount of the computing instance' resources (CPU, memory, storage, bandwidth), the estimated job duration, the current

¹<http://nui.uits.indiana.edu:8080/reports/weekly/latest/index.html#3>, last accessed on 17 Apr. 2011.

²“TOP500 list – November 2010”: www.top500.org.

³Equivalent to more than 200K single Intel CPU Core i7 965 machines.

utilization of the system and availability of the computing instances. The group quotas and technical requirements for the job are used to calculate the job's priority level and resulting schedule in the system. This rather simplified example shows the scope of decision variables a consumer has to deal with when submitting a job for computation. The group's usage information history, usage policies between institutions and groups, as well as the technical job descriptions reported for the consumer, influence the outcome of the resource manager. Thus, the efficiency of the outcome depends on the correctness of the consumer's estimated values. Economic incentives are not incorporated into allocation processes. *Moving computational services to the market demands the incorporation of price-based allocation and incentive mechanisms on top of technical schedulers.*

In this context, the paradigm shift after Grid computing is Cloud computing, which focuses on the business view of differentiated service types – infrastructure, platform and application services (Foster et al., 2008; Lenk et al., 2009). Consumers are able to scale their applications on demand with Cloud services and only pay for what they use. Studies show that businesses can reduce their total IT costs by using Cloud services to cover their peaks, instead of maintaining their own computing and application infrastructures (Mell and Grance, 2009a). Symantec (2008) reported that, on average, companies work with more than 1000 applications, most of which are web applications, as well as transactional, messaging, and collaboration applications. Current technologies like Cloud computing allow these applications to be carried out efficiently, in terms of scalability, resource utilization and costs since “using 1000 servers for one hour costs no more than using one server for 1000 hours” (Armbrust et al., 2009; Amazon, 2010b). Scalability is achieved through abstraction, distribution and virtualization techniques for infrastructures, platforms and application services. On the one hand, consumers profit from transparent, flexible and scalable services with on-demand usage and payment; on the other hand, providers profit from efficient resource management, as well as resource utilization due to economies of scale (Armbrust et al., 2009).

The pricing models most commonly offered today are *pay-per-use* and *subscription*, for which consumers pay fixed (or static) prices for a Cloud service unit, e.g. CPU-hours and GB-storage (Weinhardt et al., 2009). *Pay-per-use* is a static pricing model for service usage, whose price per time unit remains stable over time and does not depend on dynamic parameters like supply and demand. This model is typically used for products or services, which are supplied for the short term, on demand, and have similar qualitative characteristics (e.g. Amazon EC2 instances). Through the *subscription* pricing model, the consumer subscribes to use a Cloud service with a

well-defined quota on a monthly or yearly basis. Static pricing models simplify the planning of billing rates for consumers and providers, however, due to the insurance and convenience effect, consumers tend to overestimate their usage when choosing a static price tariff (Lambrecht and Skiera, 2006). In the case of resource reservation, such overestimation will produce inefficiencies in service usage. Moreover, static prices introduce unrealized utilities by consumers and providers when prices are not in keeping with the demand and supply (Lai, 2005). *Lai (2005) discusses the application of market mechanisms for the dynamic pricing of computing services, which considers fluctuations in supply and demand, thereby resulting in a more efficient allocation of computing services to consumers as compared to static pricing.* Furthermore, on-demand service usage versus guaranteed static quotas incentivize consumers to be more careful when developing their jobs or applications (Armbrust et al., 2009). In this context, *Amazon* started a *spot market* service for EC2 instances, where prices are determined based on current supply and demand (Amazon, 2009). Current snapshots show that, on average, spot prices of EC2 instances are 30% lower than the static ones.⁴ Emerging open standards, web interfaces and tools strive to overcome usage barriers to such markets by reducing switching costs and offering guarantees for service availability and data protection.

In a scenario where computing services are allocated with market mechanisms, consumers and providers need a set of tools and methodologies to interact with the market. The decision of what to bid for on-demand computing services is performed by the so-called bidding strategy, which implements the related logic of information aggregation from locally (own preferences and past experience) and publicly available sources (market information). Technical interaction with the market is facilitated through bidding agents and a well-defined communication protocol.

1.2 Research Outline

The overall goals of this thesis is to provide i) a theoretical framework for designing bidding strategies within a market-based scheduling context and ii) a technical framework for implementing bidding agents, bidding strategies and a communication protocol. The target market mechanism of this work belongs to the class of double-sided auctions addressing a scenario with multiple providers and consumers, and dynamic supply and demand (Lai, 2005). The trading objects are computing

⁴The Cloud Market provides information on actual Amazon spot prices, as well as spot price history for various computing instances, <<http://thecloudmarket.com>>.

services (or infrastructure services, IaaS), which are realistic candidates for perfect substitutes – offered by different providers, but uniformly consumable from each application based on uniform APIs. Moreover, the literature mainly focuses on the general design of auction mechanisms and bidding languages, and less on the design of bidding agents, bidding strategies and communication protocols for the field of market-based scheduling (Chevalyere et al., 2006; Parsons and Klein, 2009).

The overall goals are investigated on the basis of four main research questions, which represent the research outline of this work. The first goal (i) of this thesis is addressed in research questions 1 and 4; and the second goal (ii) is addressed in research questions 2, 3 and 4.

The first research question, RQ 1, deals with the elaboration of design characteristics for bidding strategies within the context of market-based scheduling:

Research Question 1 <Design of Bidding Strategies>

How can bidding strategies for market-based scheduling be designed and implemented, which when instantiated into bidding agents, automate the bidding process for consumers and providers?

The original term, *bidding strategy*, used in game theory refers to the set of possible actions and probability distribution function with regard to these actions at any stage of a game (Shoham and Leyton-Brown, 2009). This postulates the fact that each of the agents makes decisions based on given local or commonly shared knowledge about the actions of other agents, as well as the actual system state. Classic approaches suggest that controlled laboratory experiments with human participants should be performed when designing bidding strategies for a given market mechanism (Axelrod and Hamilton, 1981; Selten et al., 1997). Human participants are allowed to play with the market mechanism for a certain time and later asked to implement their bidding strategies in code, which reproduces their bidding behavior. In later multiple numerical experiments with continuous double auction, Das et al. (2001) showed that software agents are able to outperform humans. Moreover, software agents “don’t get distracted” or are indifferent between decisions and do not suffer from “auction fever” (Greenwald et al., 2003; Ku et al., 2005).

Recent research shows that bidding strategies are implemented as complex algorithms with multiple functional steps, which aggregate data according to individual demand functions, experience from past actions and shared market information of other agents’ actions (Parsons and Klein, 2009). Moreover, a major part of the literature elaborates on bidding strategies in the context of financial markets utilizing scoring functions for monetary profit maximization, however, there is a lack of

research for bidding strategies in the context of market-based scheduling with applications or computing services for specific scoring functions, which consider additional metrics like makespan (Parsons and Klein, 2009; Reeves et al., 2005; Heydenreich et al., 2010).

A bidding strategy executes a policy, a so called scoring function, which formally defines the goals and constraints that govern the decisions. “Even a goal as superficially simple as *maximize utility* will require a human to express a complicated multi-attribute utility function” (Kephart and Chess, 2003). A common evaluation methodology for bidding strategies is their instantiation into software agents and the execution of numerical experiments in an agent-based environment (MacKie-Mason and Wellman, 2006; Tesfatsion, 2006). A detailed elaboration of an appropriate evaluation methodology of complex strategies is part of RQ 4.

While RQ 1 focuses on the game theoretic design of bidding strategies, the second research question, RQ 2, deals with the technical design and realization of software agents and bidding strategies within a general software framework with well-defined interfaces and methodology:

Research Question 2 <Design of a Framework for Automated Bidding>

What are the characteristics of bidding agents and how can they coincide with bidding strategies in an agent framework for market-based scheduling?

A system for market-based scheduling is distributed and contains the market middleware with related components for running auctions, contract management, enforcement of service level agreements and security, as well as the consumer and provider tools to interact with the market and execute consumer applications on the provider’s computing services (Nimis et al., 2008). A “grand challenge” of such complex systems is to make them self managing, since many components implement their own logic and functionality and run “beyond company boundaries into the Internet” (Kephart and Chess, 2003). Kephart and Chess (2003) identified that such systems have to be designed to run autonomously since rising complexity “appears to be approaching the limits of human capability” “for even the most skilled system integrators to install, configure, optimize, maintain, and merge.” The basic entity of systems’ components are agents, which interact autonomously according to high-level objectives, set by their owners or administrators. Moreover, each of the autonomous components is “responsible for managing its own internal state and behavior and for managing its interactions” with the related components in the system (Kephart and Chess, 2003). Therefore, a system for market-based scheduling will, in fact, be a multi-agent system built with commonly accepted principles and communication protocols.

In order to interact with the market mechanism, consumers and providers need bidding tools, which coordinate and execute the bidding processes autonomously, and integrate with the heterogeneous system through well-defined interfaces and communication protocols. The here developed *Framework for Automated Bidding* called *Bid-Generator* provides interfaces and a methodology for implementing bidding strategies and bidding agents. Existing agent frameworks are either generic or domain specific, e.g., for realizing mobile agents, robots, search engines and bidding agents for financial markets (known also as algorithmic traders) (Jennings et al., 1998; Bergenti and Poggi, 2002; Sturm and Shehory, 2004). Moreover, there are existing agent frameworks, which are either designed for experimental purposes or tournaments (Wellman et al., 2007; Cai et al., 2009). However, there is a lack of research in the area of agent frameworks in the domain of market-based scheduling and its characteristics, which is the area that RQ 2 investigates. Common evaluation methodologies of software frameworks are based on functionality comparison, proof-of-concept implementation or integration in running systems according to the investigated use cases (Lind, 2001; Zambonelli et al., 2003; Sturm and Shehory, 2004; Bartolini et al., 2005). Such integrated systems are often evaluated with performance tests.

A distributed system for market-based scheduling interacts by exchanging messages over well-defined (Web service) interfaces. The aim of the third research question, RQ 3 is the specification and the realization of protocols for the exchange of context-specific messages between the bidding tools and related market components:

Research Question 3 <Communication Protocols>

What are the characteristics of a message exchange within a market-based scheduling context? How can technical and economic preferences be expressed, communicated and matched between consumers, providers and the market?

The communication in a distributed system for market-based scheduling is ruled according to a well-defined protocol with specific information content between the different components and communication directions, which is referred to here as a communication protocol. For example, bidding strategies generate bids of economic and technical attributes in the form of multi-attributive messages, which are submitted through the bidding agents to the market. The market returns multi-attributive match messages as well as market information on the actions of other agents. The protocol specifies the types of messages exchanged between specific components – application and bidding agent, market and bidding agent, etc. There is a need for formal languages that enable the expression of economic and technical information (multi-attributive), where the language concepts, their properties and relations are

commonly accepted to create an ontology (Kephart and Chess, 2003). The economic and technical information represents the needs and preferences of consumers and providers regarding the demanded and traded computing service. System developers “will need tools that help them acquire and represent policies – high-level specifications of goals and constraints, typically represented as rules or utility functions” (Kephart and Chess, 2003). The economic and technical information communicated in the form of bids and the resulting matches are not only used to execute consumer applications to the allocated provider’s computing service, but controlling system components use it to monitor and enforce the agreements (market match of legally binding consumer and provider bids) as a result of the matchmaking process.

A semantic specification of bidding languages is crucial in a distributed system since the related components, the consumer’s applications and provider computing services can span across different administrative domains – on premiss, as well as off premise. This will enable more comprehensive usage and trust, and increase the validation and verification capabilities of the controlling components (Kephart and Chess, 2003). Furthermore, consumers and providers can easily reason about the posted bids and their technical descriptions in the market, as well as derive better knowledge with regard to the supply and demand of computing services.

Available communication protocols for auctions are often proprietary (e.g., eBay, Amazon Web Services), consisting only of technical attributes (e.g., Job Submission and Description Language) or developed for tournaments, such as the *Trading Agent Competition* (Anjomshoaa et al., 2005; Niu et al., 2009). There is a lack of research on communication protocols that also express economic data to the technical attributes and in a market-based scheduling context (Andreetto et al., 2010; Laure et al., 2006; Smirnova, 2009). Similarly to RQ 2, the evaluation methodology of communication protocols (as part of a distributed system) is analytical and combined with proof-of-concept implementation and performance tests.

The fourth research question, RQ 4, refers to the evaluation of complex bidding strategies in homogeneous and heterogenous settings, and also contains the proof-of-concept evaluation of the *Framework for Automated Bidding* and communication protocol developed here.

Research Question 4 <Evaluation of Bidding Strategies>

How do learning-based bidding strategies score against benchmark bidding strategies in settings with homogeneous and heterogeneous agents?

There is no established and clear methodology for evaluating complex bidding strategies and their interactions in markets. This has to do with the target use cases

evaluated, the assumptions selected, as well as the scenarios and types of market mechanisms analyzed. Common assumptions in classic game theory are that agents are rational, perfectly informed and apply bidding strategies that are the best response to the actions of other agents. Furthermore, the mathematical model of the bidding strategies is similar for all agents, thus, they are often assumed to be homogeneous (Shi and Jennings, 2010). In game theory, finding the Nash equilibrium is one of the main goals in analyzing bidding strategies (Shi and Jennings, 2010; Wellman, 2006; MacKie-Mason and Wellman, 2006). As a system grows in complexity, game theoretic models quickly become unfeasible. This is a reflection of the “strategies space, number of agents, degree of incomplete and imperfect information, and dynamism” (MacKie-Mason and Wellman, 2006; Tesfatsion, 2006). In realistic settings, the number of consumer and provider agents is asymmetric, not all agents share the same information, agents do not share the same preferences as well as demand and supply is dynamic over the time. In respect to their use cases – e.g. interactive applications or batch jobs – consumers are interested for immediate or delayed purchases of online computing services. Thus, realistic settings require market mechanisms with continuous matching and allocation of computing services (e.g., *spot market*⁵) or markets for short-time future contracts for their batch jobs (*call market*⁶). In online settings, demand and supply fluctuates over time and agents are continuously making decisions based on the available information about the other agents’ actions. Computing Nash equilibrium is an NP-complete problem and feasible in perfect information market scenarios with a small number of agents and manageable action space.

In order to evaluate bidding strategies for realistic online scenarios, extensive experiments in homogeneous and heterogeneous settings need to be carried out. The research in this direction is still ongoing and generally addresses financial markets scenarios than trading computing services (Das et al., 2001; Tesauro and Das, 2001; Vytelingum et al., 2008; Phelps et al., 2010a). A commonly applied market mechanism in these settings is the *continuous double auction*. Moreover, these bidding strategies are developed in the context of financial markets with common goal of profit maximization. In the context of market-based scheduling there is a lack of research in appropriate bidding strategies with sophisticated scoring functions, e.g. “maximize profit and utilization,” “minimize time to complete and payments” or

⁵A market in which a commodity is bought or sold for immediate delivery or delivery in the very near future, <<http://financial-dictionary.thefreedictionary.com/Spot+market>>.

⁶A market in which trading in individual securities occurs at specific times as opposed to continuously, <<http://financial-dictionary.thefreedictionary.com/Call+Market>>.

other important performance indicators of a technical or economic nature (Reeves et al., 2005; Heydenreich et al., 2010). Furthermore, evaluation of heterogeneous agents, which apply different bidding strategies in competing online market settings, is partially explored in the existing research, and virtually unexplored in the context of market-based scheduling. Therefore, RQ 4 evaluates complex settings of markets and agents' bidding strategies; and the proof-of-concept implementation shows the feasibility of the models and methodologies developed here (MacKie-Mason and Wellman, 2006; Tesfatsion, 2006; Sturm and Shehory, 2004; Bartolini et al., 2005). Moreover, the models presented in this work are implemented, integrated and tested in a real project according to real use cases for batch and interactive applications (Nimis et al., 2008, 2009; Neumann et al., 2007).

1.3 Summary of the Research Contributions

This work contributes to the research questions introduced by providing novel theoretical and software models – a novel bidding strategy, called Q-Strategy, a software framework for realizing both bidding agents and bidding strategies, called BidGenerator and a communication protocol for expressing and exchanging economic and technical information within a system for market-based scheduling, called *MX/CS* (*Message Exchange in Computing Service Markets*). The contributions of this work can be summarized in the following way:

Development and realization of a novel, adaptive and configurable bidding strategy called Q-Strategy: The first contribution of this work, the *Q-Strategy*, is a novel model for structuring trading activities for trading objects in terms of related consumer and provider requests, their bids, and observed rewards. Consumer and provider requests consist of multi-attributive technical and economic preferences for each of their applications or computing services. Furthermore, each of the trading objects can be associated with an own scoring function. Thus, the *Q-Strategy* aims to solve a so-called *multi-armed bandit problem*, where each of the arms represents a long-term optimization problem for a given type of trading object (Borissov and Wirström, 2008; Borissov, 2009; Borissov et al., 2010). Section 3.4 presents the design model of the *Q-Strategy*, its realization methodology and integration into a market-based scheduling scenario.

Development and realization of a novel agent framework for automated bidding called BidGenerator: The second contribution of this work, the *BidGenerator* framework, offers well-defined interfaces and a methodology for implementing

any kind of bidding agent and bidding strategy. As a proof-of-concept, it contains implementations of several state-of-the-art bidding strategies, as well as the *Q-Strategy*. Based on consumer and provider preferences, as well as the target market mechanism, bidding agents can be associated with different adequate bidding strategies from the available pool (Borissov et al., 2010). Each of the bidding agents connects to the SORMA market over well-defined interfaces and exchanges secured and binding market messages. The architecture of the *BidGenerator* framework is presented in Section 4.4.

Development and realization of a communication protocol for market-based scheduling called MX/CS: As part of the distributed system for market-based scheduling, the third contribution of this work is the definition of a communication protocol, which defines the type of information exchanged between related system components in the message chain, from the application request to the bidding agent, from the bidding agent to the target market, and from the market back to the application. As proof-of-concept, the communication protocol has been developed, integrated and tested on top of the Job Submission and Description Language (Borissov et al., 2009b). A detailed presentation of the communication protocol can be found in Section 5.3.

Development of a methodology for evaluating bidding strategies in complex settings: This work provides a full-fledged analysis of bidding strategies in homogeneous and heterogeneous settings using recognized methodologies for agent-based simulations. The lessons learned from the derived evaluation scenarios, their iterative evaluation over the years and the search for an appropriate evaluation methodology has led to the development of a more detailed methodology for agent-based experiments, which is presented in Section 6.1. As part of the fourth contribution of this work, the developed methodology was applied by evaluating the *Q-Strategy* against benchmark strategies in homogeneous and heterogeneous settings in a *spot market*. Furthermore, the evaluation results in Section 6.2 show that the *Q-Strategy* outperforms the benchmark strategies in most of the scenarios and elaborate the cases in which the benchmarks were more successful than the *Q-Strategy*.

Proof-of-concept implementation of BidGenerator, the Q-Strategy and MX/CS in a real running prototype for market-based scheduling as part of the SORMA project:⁷ The fifth contribution of this work is the proof-of-concept implementation, integration and successful testing of the *BidGenerator* framework

⁷<http://sorma-project.eu/>

with the *Q-Strategy* and the *MX/CS* communication protocol in a real running prototype for market-based scheduling (Nimis et al., 2008, 2009; Neumann et al., 2007). The integrated prototype has been evaluated according to two case studies, which execute batch and interactive applications through distributed providers of computing services. The technical analysis of integrations with *BidGenerator* and *MX/CS* is showed in Chapter 7. Performance experiments demonstrated the efficient proof-of-concept implementation and integration with the related core components of the overall prototype.

1.4 Structure of This Work

Figure 1.1 depicts the overall structure of the thesis. Chapter 1 introduces the topic, research questions and contributions of this work. Chapter 2 discusses the target domain of this research and specifies the economic and technical environments of the market-based scheduling system. The economic environment describes the microeconomic model of the system, a definition of the transaction object traded and challenges when designing markets for scheduling computing services. The technical environment introduces two application scenarios for batch and interactive jobs, describes the realization challenges of the system for market-based scheduling from technical perspective, depicts the technical architecture of the system and introduces the target bidding scenario. It concludes with a presentation of the research methodologies applied in the evaluation of the economic and technical models developed.

Chapter 3 elaborates on the field of bidding strategy design. It starts with the definition of the term *bidding strategy* and the specification of related design desiderata from a *computational mechanism design* perspective. Subsequently, existing non-adaptive and adaptive bidding strategies are investigated and evaluated according to the design desiderata derived and applied in a market-based scheduling domain. Finally, a novel adaptive bidding strategy called the *Q-Strategy* is introduced as part of RQ 1.

Chapter 4 presents the design and realization of a *framework for automated bidding* called *BidGenerator*. The chapter starts with a discussion of what a *bidding agent* is. Design desiderata for developing bidding agents are derived; based on these, existing agent frameworks are elaborated and evaluated. As a contribution to RQ 2, the chapter concludes with the presentation of the *BidGenerator* framework.

Chapter 5 focuses on the design and realization of a communication protocol for market-based scheduling. The chapter starts with specification of the design desider-

Part I Foundations	Chapter 1 Introduction		
	Chapter 2 Preliminaries and Related Work		
Part II Design and Implementation	Chapter 3 Economic Design of Strategies for Market- Based Scheduling	Chapter 4 Architectural Design of a Framework for Automated Bidding	Chapter 5 Communication Protocols in Computing Service Markets
	Chapter 6 Agent-Based Numerical Experiments		
	Chapter 7 Technical Analysis and Application		
Part III Evaluation			
Part IV Conclusion	Chapter 8 Summary of This Work and Future Research		

Figure 1.1: Structure of this work

ata for developing communication protocols in the target domain. Subsequently, it presents and evaluates related works on existing communication protocols to the design desiderata. The contribution to RQ 3 is a formal definition of a novel *communication protocol* called *MX/CS*, specifically designed for a market-based scheduling domain.

Chapter 6 presents the results of the agent-based experiments performed – the evaluation of the *Q-Strategy* against benchmark bidding strategies in homogeneous and heterogeneous settings. The chapter starts with a description of the evaluation methodology and presents the design of the numerical experiments. The contribution to RQ 4 is an insightful presentation of the results on the impact of the learning-based adaptive bidding strategy, *Q-Strategy*, against benchmark bidding strategies in a spot market for computing services, realized with the *continuous double auction*.

Chapter 7 is a technical analysis of the *BidGenerator* framework and *MX/CS* applied in real case studies. Two *case studies* for *batch* and *interactive applications* show how *BidGenerator* and *MX/CS* are applied for automating provisioning and acquisition processes in markets for computing services. A third *case study* shows how *MX/CS* is applied in the creation of electronic contracts. In this chapter, the adoption of *MX/CS* in an application scenario for sharing storage services in social networks

called *Social Cloud* is also described. The chapter is concluded with a performance analysis of the integrated system of *BidGenerator*, *MX/CS* and market components. Finally, Chapter 8 concludes this thesis with a summary of the research contributions and an overview of related future research topics.

1.5 Related Publications

The preliminary ideas for the *Q-Strategy* were first presented in Borissov (2009) and Borissov and Wirström (2008), including its definition, realization and primal evaluation. The evaluation compares aggregated consumer utilities in scenarios with symmetric and asymmetric agents for specific market mechanisms. Borissov et al. (2010) present a comprehensive view on the *BidGenerator* framework and provides a fully-fledged evaluation of the *Q-Strategy* in homogeneous and heterogeneous settings.

Borissov et al. (2009b) present a specification for a bidding language called EJS DL (in this work renamed to *MX/CS*), for expressing and exchanging economic preferences in the context of market-based scheduling. A detailed description of such a bidding language is shown with a corresponding realization scenario for exchanging private, public and market data between the actors in the environment, such as consumers, providers, bidding agents and auctions. The bidding language is associated with a corresponding ontology, which defines the semantics of concepts and relations utilized with links to recognized upper ontologies.

Developed within the scope of the SORMA project, Borissov et al. (2008b); Nimis et al. (2009, 2008); Neumann et al. (2007) describe the proof-of-concept implementation and integration of the *BidGenerator* framework and *MX/CS* in a real prototype.

Chapter 2

Preliminaries and Related Work

THE aim of this chapter is to introduce the research field of market-based scheduling of computing services. The chapter begins with a comparison of emerging service paradigms with the aim of deriving tradable computing service objects. It motivates the application of market mechanisms, which can improve the efficiency of provisioning and usage of computing services. The application of market mechanisms imply economic and technical challenges for the design of mechanisms and strategies, as well as for the design of related software tools like agent frameworks. The economic and technical challenges are discussed in the subsequent sections, followed by an analysis of existing works in the research field investigated. Finally, the chapter concludes with a discussion of the research methods available, in order to select an appropriate evaluation methodology for the contributions of this work.

2.1 Computing Services

The computer became the working unit in society for executing a multitude of tasks from simple data processing and storage to complex processing of simulations, as well as sensor and enterprise data. To execute these tasks, different computing services are often required. For personal uses, a single computing instance is needed for document creation and editing, audio and video streaming and data aggregation. Industrial and research applications in the form of information systems often produce big logs from GB to PB of data, which can only be processed in a time-efficient manner with a bundle of computing instances that can work together.

Application services became complex, connecting hundreds to millions of people, are distributed over the network and scalable. Such applications aggregate and provide data from various sources like sensors, audio and video, finance and forecasting services. Each of these applications has varying technical requirements for scalability, performance and security. The following sections briefly introduce actual computing paradigms, discussing the shift from utility computing to Cloud computing.

2.1.1 Utility Computing

The first ideas to provide computing power as a utility (similar to energy, water and communications) had been discussed back in the 60s (Armbrust et al., 2009). However, it really started to gain attention in the 90s, together with the increase in computing usage and Internet technologies in everyday life. Through the restructuring of existing computing infrastructures, utility computing aims to provide computing power in a more flexible and effective way, in order to reduce IT costs and improve provider profits. In Rappa (2004), the following characteristics of utility computing are identified:

- *Necessity.* Utility computing offers a certain value for the consumers to fulfill their day-to-day needs.
- *Reliability.* The utility computing services provided must be readily available anytime and anywhere.
- *Usability.* Utility computing services have to be simple at the point of use (“plug and play”) and hide the system complexity from the consumer. This consideration implies the existence of standards for technical access (interfaces) and message protocols.
- *Utilization rates.* Providers have economic interests in maintaining higher utilization rates and increasing profits for their computing infrastructures. Their utilization rates may fluctuate over time and across geographic regions, which have to be taken into consideration in providers’ business models.
- *Pay-per-use.* This pricing method stipulates that prices of utility computing services are calculated according to their actual usage. The prices of utility computing services can vary based on actual system utilization, time of day, geographic region and consumer type, i.e., price and client discrimination policies (Püschel et al., 2007). Providers can offer discounts during low peak times in order to shift consumer demand from higher peak times.
- *Scalability.* The scalability property of utility computing services refers to the fact that, on the one hand, consumers can get as much capacity as they are willing to pay for; on the other hand, providers can benefit from economies of scale by reducing the costs per unit when consumer usage increases.

- *Service exclusivity.* Service exclusivity represents the set of conditions and usage policies for utility computing services. The conditions are specified in the form of provisioning and usage policies, which are usually set by providers, however, they often have to be in keeping with policies of common regulative instance (e.g., government).

The characteristics of utility computing mentioned above imply challenges of a technical, business and political nature. Related processes of utility computing like repetitive tasks, service provisioning and usage, monitoring, payment and billing are not manually manageable (Kephart and Chess, 2003). Well-defined interfaces and methodologies, as well as tools, are required in order to automate these processes on the behalf of the provider and simplify access for the consumer.

2.1.2 Grid Computing

The idea of Grid computing originated from *metacomputing* projects like *I-WAY*, where *metacomputing* denotes “a networked virtual supercomputer, constructed dynamically from geographically distributed resources linked by high-speed networks.” *I-WAY* was one of the first projects, which connected supercomputers from seventeen different sites, mainly in the USA, using preliminary versions of the *Globus Toolkit* Grid middleware (Foster and Kesselman, 1997). The Grid computing paradigm incorporates ideas and relates them to paradigms like distributed and cluster computing.

Distributed computing is a computing paradigm, in which software applications are executed on one or more computing instances (also called computing nodes), which are connected and communicate through a computer network. To achieve their scalability, the software applications are usually modularized and executed on multiple computing instances in parallel (Naor and Stockmeyer, 1993). The general advantage of distributed systems over centralized systems is the ability to enable modularization of complex systems, where each of the modules are executed on different nodes connected through a network. Therefore, distributed applications can be more easily configured to avoid a single-point of failure, to improve scalability and fault tolerance through the redundancy of application modules.

A *cluster* contains a set of locally installed and independent compute instances (also called nodes) that are interconnected through a dedicated network. All components of a cluster system belong to a single administrative domain and usually reside in a single room (Baker and Buyya, 1999). A cluster system usually consists of homogeneous computing nodes, installed with the same operating system and software. In order to

submit their applications, the users connect to a so-called *user interface* (also called *head node* or UI node) in order to interact with *Cluster Management Systems* (CMS). One of the main CMS components is the *scheduler*, which uses well-defined policies to allocate submitted applications to *computing nodes*.

In contrast to distributed and cluster computing, a Grid computing system spans across multiple sites, each of them coordinating and sharing their own computing service instances to the research community. A global resource scheduler monitors the state of each of the registered sites in terms of the current site utilization, as well as the number of running and allocated jobs. The decision to allocate jobs to sites is made by the global resource scheduler according to the job's technical requirements, the assigned Virtual Organization¹ (VO) of the job owner and the utilization of the target Grid sites assigned to the owner's VO. The local scheduler of the allocated Grid site performs the final scheduling actions. An overview of common scheduling mechanisms for Grid systems is presented by Dong and Akl (2006).

According to the *three point checklist*, a Grid i) “coordinates resources that are not subject to centralized control,” ii) “uses standard, open, general-purpose protocols and interfaces,” and iii) “delivers non-trivial qualities of service” (Foster, 2002). Grid systems have been in place for a decade mainly in the field of research. Prominent projects like *TeraGrid*,² *Open Science Grid*³ and *EGEE*⁴ (Enabling Grids for E-scienceE) already connect many sites around the world, enabling the calculation of huge amounts of data, which is generated by experiments in areas like physics, biology and chemistry. With more than 80K CPU cores, distributed in around 250 computing centers with more than 9K users, the EGEE Grid handles more than *200K jobs each day* (Lingrand et al., 2009).

In order to integrate, provide and perform Grid computing, one needs well-defined interfaces and standards that are supported by each provider (e.g., through common Grid service management tools), in order to prepare computing services for the Grid. Grid standards are mutually created and approved by global standardization entities like the *Open Grid Forum*⁵ (OGF) and the *Organization for the Advancement of Structured Information Standards*⁶ (OASIS). The *Open Grid Services Architecture*

¹The “project community” of a consumer or provider is called *Virtual Organization* or VO. A VO can contain Grid services of many geographically distributed Grid sites.

²<<https://www.teragrid.org>>, last visited on 15 Mar. 2010.

³<<http://www.opensciencegrid.org>>, last visited on 15 Mar. 2010.

⁴<<http://www.eu-egge.org>>, last visited on 15 Mar. 2010.

⁵<<http://www.ogf.org>>, last visited on 15 Mar. 2010.

⁶<<http://www.oasis-open.org/home/index.php>>, last visited on 15 Mar. 2010.

(OGSA) is a reference model for implementing Grid systems, which defines its concepts, components and relations (Foster et al., 2005). The realization of Grid services is based on open standards, which is why the Web service approach was selected. The original Web service approach supports only stateless communication. Based on a decentralized control of the Grid, there was a need for new protocols on top of the Web service protocols, which enable the management of Grid service states. Such protocols have been defined within the *Open Grid Services Infrastructure* (OGSI) document and later refined and implemented as the *Web Services Resource Framework* (Foster et al., 2005). The *Simple API for Grid Applications* (SAGA) provides a common API for application submission and management, data management and monitoring facilities (Goodale et al., 2008). This API is designed to work with and support most available Grid middlewares through connectors. The most prominent Grid middlewares are the *Globus Toolkit*, *gLite* and *UNICORE* (Ellert et al., 2007). A Grid middleware offers software packages, which enable the provider to connect their computing instances to the Grid as well as for development, deployment and management tasks, such as Grid service deployment, monitoring, discovery, management and security. For consumers, the Grid middleware provides the necessary APIs to prepare their applications for the Grid, as well as tools for application management, monitoring and communication with the Grid system.

Consumers willing to utilize Grid services need to prepare their application according to the APIs of the target Grid middleware. The application's technical (hardware and software) requirements, as well as application specific attributes (e.g., for application execution and data staging) are expressed with a well-defined description language like the *Job Submission and Description Language*(JSDL) (Ellert et al., 2007). The Grid scheduler acts as a central broker and schedules the consumer's applications to Grid sites according to the globally available Grid site information, the associated consumer's VOs and the technical requirements of the applications. The current achievements in Grid computing are the ability to connect geographically distributed grid sites and manage VOs, enabling a distributed and managed execution of applications across the available Grid services of the consumer's VO.

Like with utility computing, the provisioning and usage of Grid services should make current infrastructures technically and economically more efficient and sustainable. As such, the so-called *Grid Economy* (Buyya et al., 2005) discusses market models for offering Grid services as commodities through negotiation or auctioning processes between consumers and providers. Economic efficiency refers to the fact that market mechanisms for computing services can take the technical and economic preferences of consumers and providers into consideration and match them more efficiently than

current technical schedulers and static prices. On the one hand, consumers execute different applications with varying requirements, depending on application type, as well as time and budget constraints. On the other hand, providers want to achieve higher utilization and profits with their computing infrastructures (Dong and Akl, 2006). To achieve this, the *Grid Economy* has to offer appropriate market mechanisms, bidding tools and communication protocols, which support the market allocation and bidding processes of consumers and providers. The literature on Grid economics has become a significant milestone for the next wave to move the Grid vision to business and expand it with new types of services such as Cloud computing.

2.1.3 Cloud Computing

Cloud computing builds on the ideas of Grid computing, but focuses more on the business view of differentiated service types – infrastructure, platform and application services. Since the Cloud computing hype has started, the term has been defined differently by experts and market analysts from research and industry (Foster et al., 2008; Geelan, 2009; Mell and Grance, 2009b; Plummer et al., 2008). Furthermore, several joint efforts have been started to define a common view of the characteristics, the related technical and economic challenges of the Cloud stack and to work on standardization of the related APIs. Organizations like the *Open Grid Forum*, Distributed Management Task Force’s Cloud Incubator (DMTF, 2010a), the *Open Science Grid* and the *Open Cloud Manifesto* (Nelson, 2009) already include representatives from research and industry, however, this research field is just opening up and many challenges including, but not limited to security, interoperability and economics have to be elaborated.

The general idea shift from Grid to Cloud computing is provided in the following definition (Foster et al., 2008):

Definition 2.1.1 (Cloud computing). *A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically scalable, managed computing power, storage, platforms, and services are delivered on-demand to external customers over the Internet.*

The key points of the definition are that Cloud computing services i) scale based on consumer demand, which allows for efficiency through economies of scale, where ii) scalability is achieved through abstraction and virtualization over the three Cloud computing service types – infrastructure, platform and application services, and iii)

the Cloud computing services are dynamically configured and delivered on demand, where iv) the consumer pays for usage time.

A more detailed definition of Cloud computing is provided by Mell and Grance (2009b), which fundamentally contains the definition of Foster et al. (2008), but extends it by describing the three Cloud service types – *Infrastructure as a Service*, *Platform as a Service* and *Software as a Service* (Figure 2.1) – as well as the four possible deployment models of the Cloud computing paradigm – *public*, *private*, *community* and *hybrid Clouds*. The following sections will look into more details of the *Infrastructure as a Service* model, which is the focus of this research, as well as its deployment types and characteristics.

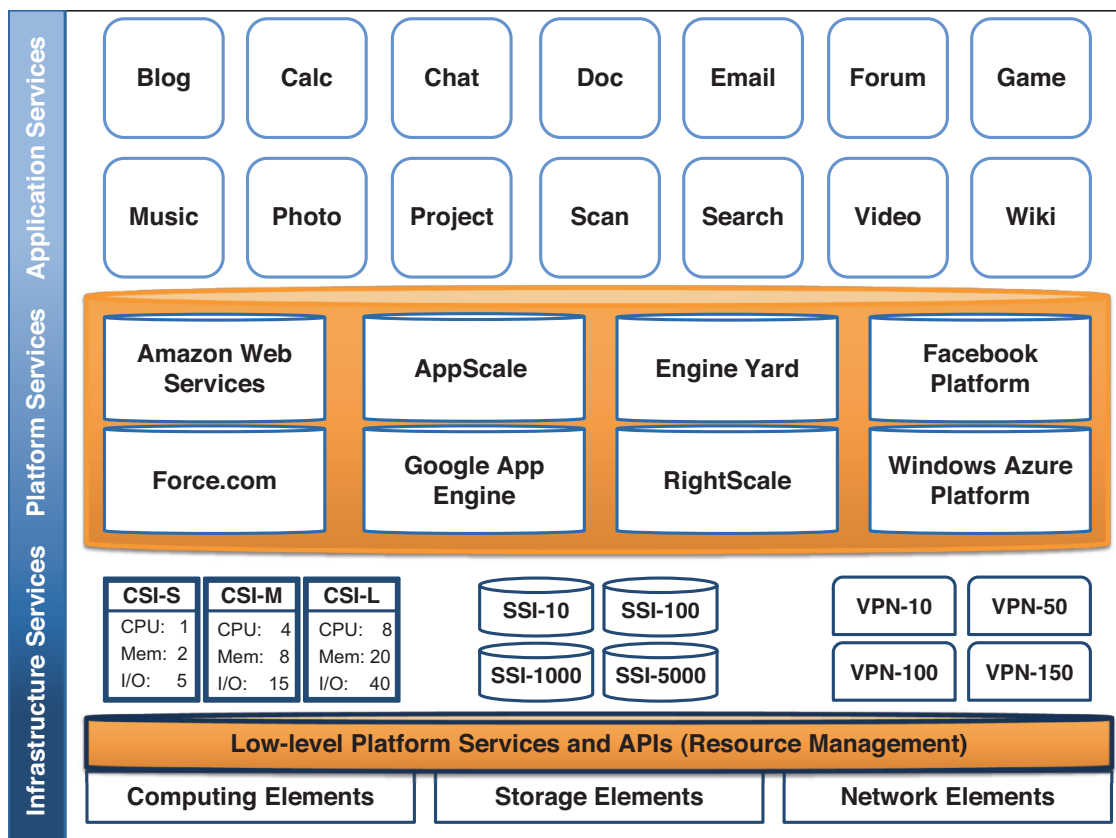


Figure 2.1: Cloud services (own representation)

2.1.3.1 Infrastructure as a Service

Towards the shift of Cloud computing, computing centers aim to organize and coordinate their computing resources (computing nodes, storage nodes, network bandwidth, etc.) more efficiently, in order to provide scalable computing power on demand as well as to identify idle capacities. Such free capacities can be offered for public use in order to increase the provider's profit.

Most providers of infrastructure services adopt static pricing models like *subscription* and *pay-per-use*, where the price for a service instance is fixed in the time (Weinhardt et al., 2009; Clemons, 2009). Static prices do not reflect changing market conditions in supply and demand, therefore providers' computing infrastructures have idle capacities. In order to respond to this issue, *Amazon* started a *spot market*⁷ where free capacity is allocated dynamically based on the current supply and demand. Prices in the spot market vary dynamically (Amazon, 2009).

In the context of Cloud computing, *Infrastructure as a Service* (IaaS) refers to the type of fundamental computing services running on top of hardware infrastructures, such as bandwidth (network), computing, storage and database services (definition of *Resource* in Treadwell (2007); Mell and Grance (2009b)). In the case of purchasing a computing service, a consumer has controlled access to it, which enables deployment of the target execution environment in terms of software dependencies and environmental settings (e.g., environment variables and network security settings).

Definition 2.1.2 (Computing Service Instance). *A computing service instance (CSI) provides a **computing environment** that enables the execution of applications by satisfying their technical requirements for a specified number of processors, main memory, storage, bandwidth and operating system. The functionality to be able to access a CSI in terms of transferring and executing an application, as well as retrieving the related result is enabled through well-defined APIs (Treadwell, 2007). The exact description of the **application execution plan** in terms of usage, duration, payment methods, deployment, execution and result retrieval is provided in a well-defined message protocol (i.e., an application description language). The **application execution plan** is locally performed by the **runtime environment** of the computing service. A well-defined message protocol refers to a standardized means of describing applications's requirements, CSIs and their properties.*

⁷A market in which a commodity is bought or sold for immediate delivery or delivery in the very near future.

Based on this definition, a provider of *IaaS* instances enables the transfer and execution of applications on its CSIs, which can be physical or virtual machines (Nurmi et al., 2009). Consumers receive the necessary access control to communicate with the provider *IaaS* infrastructure and applies the related tools to prepare the target environment and their applications in order to be executed on the allocated CSI. Dependencies on third party software components and related licenses are optional services, which can be additionally purchased by the *IaaS* provider or consumers can transmit them together with their applications. In the end, when the application is completed and results transferred, the computing service is released and reverted to its initial state.

According to Foster et al. (2006), *Infrastructure Services* consist of several management units – execution management services, data services, resource management services, security services and monitoring services, which enable management of the *IaaS-Platform*:

- *Execution Management Services*. The execution management (also called runtime management) controls the starting, execution and termination processes of a deployed application (Papazoglou and van den Heuvel, 2007). In the Grid computing context, the Globus Resource Allocation Manager (GRAM) is the interface service for submitting, locating, monitoring and canceling computing jobs (Foster et al., 2008). The GRAM also supports the JSDL format, which consists of elements to specify the runtime scripts and basic configurations of jobs in order to start them on the allocated CSI. Similar, providers of *IaaS*-services, which are offered over a market mechanism, need to rely on standardized interface implementations in order to enable common deployment, execution and monitoring of batch and interactive jobs.
- *Data Services*. Applications have different requirements with regards to data management, data usage and persistence. An *IaaS* platform has to offer flexible facilities (APIs) to bind and use storage services to facilitate the requirements of applications like *Document Management*, *Indexing*, *Logging* and *Memory Management* (Buyya et al., 2009).
- *Messaging*. A crucial part of a computing platform are the message protocols that enable and automate communication with the *IaaS* services, such as submitting messages to deploy and start a specific application, but also for retrieving (monitoring) runtime specific data of the executed application, or execute payments for usage. Such message protocols are typically encoded in an

XML (or a JSON) message format and are communicated through Web service interfaces (Foster et al., 2009). Standardization plays a crucial role in achieving interoperability between computing systems with well-designed, useful and well-accepted message protocols. Standardization bodies like the W3C, OASIS, DMTF and OGF review and publish various well-defined and commonly applied message protocols in fields like Web service communication and security protocols, business process management, job submission and execution (e.g., JSDL), to mention just a few. In the context of this work, bidding languages are a further part of the message protocols of an *IaaS* platform enhanced with market-based scheduling facilities (Nisan, 2006).

- *Resource Management.* Resource management facilitates the execution of *IaaS* services on the underlying resource infrastructure of computing nodes, storage elements and network services. Scalability of an application in *IaaS* can be achieved through modularization and parallel execution on more than one node (Yu and Vahdat, 2006). In the context of this research, consumers are able to acquire as many CSIs as needed through the computing services market to run their batch or interactive applications.
- *Security Services.* The security module handles the processes and methodologies, which are related to authorization, authentication, secure communication and data security (Foster et al., 2002; Rittinghouse, 2009; Catteddu and Hogben, 2009). In the context of this work, the communication of consumers, providers and the market has to be authorized and secured. As a part of the market model and scenario, all submitted consumer or provider bids are binding, i.e., the bidders are responsible for using and providing the CSIs as specified in the bids and matched (a match results in a binding contract) by the market. In the context of the SORMA project, the authorization and signature of bids was realized with the emerging standard for single sign-on authentication – Security Assertion Markup Language (SAML) (Armando et al., 2008; Nimis et al., 2009).
- *Monitoring.* Monitoring is a significant part of the administration of an infrastructure service, which enables controlling the system by measuring for system failures either proactively or reactively and thus preserving the reliability of the *IaaS* platform. Monitoring services provide performance data, utilization statistics, response times, network load statistics, transaction statistics, load balancing, health management and troubleshooting (Bernstein, 1996; Foster

et al., 2006; Papazoglou and van den Heuvel, 2007). *IaaS* providers use the monitoring data of applications for auditing and billing purposes. The storing and analysis of such data has to be in compliance with the legal stipulations of the institutional policies (e.g., governments) (Mowbray, 2009). In the context of project SORMA, the monitoring services provided runtime information of the CSIs to the *service level agreement enforcement* and *contract management* components (Nimis et al., 2009).

- *Contract and Payment.* A contract is a legal and binding agreement between a consumer and a provider that captures the already negotiated technical and economic objectives, which are also called *service level objectives* (SLOs) are part of a *service level agreement* document. To ensure regular execution of the contract, SLOs are frequently monitored by the target provider and consumer. They are the main indicators for measuring the success of a contract and the resulting payments or penalties. SLOs can be assigned any quality level of service attributes (e.g., “service availability of 99.95%”), technical attributes (e.g., number of CPUs), payment procedures, penalties or legal aspects (Becker et al., 2008; Wilkes, 2008; Papazoglou and van den Heuvel, 2007). To provide this capability, the *IaaS* platform should ensure that consumer applications are executed according to contract and react adequately on irregularities. The irregularities are identified by periodic monitoring of the SLOs. The payment procedure contains information regarding when and how payments are to be transferred from the consumer to the provider. So far, current payment transactions are often executed via credit card or online payment services like *PayPal*. Online payment services often provide Web service interface, which allows automation of the payment process (Armbrust et al., 2009). In the context of SORMA, price, technical requirements, SLOs, penalties, method of payment and establishing contracts are part of the message protocols and the related bidding processes of consumers and providers, whose aim is to reach a technically and economically matching agreement in an auction setting (Borissov et al., 2009b; Nimis et al., 2009).

2.1.3.2 Deployment Models

Depending on the organizational structure, there are four models for deploying Cloud services, which can be separated into four main models – *Private Cloud*, *Community Cloud* and *Public Cloud*, as well as a mixed model called *Hybrid Cloud* (Mell and Grance, 2009b; ISACA, 2009; Microsystems, 2009; Craig et al., 2009).

Table 2.1 summarizes the three main models of Cloud service deployment and their key characteristics. The three deployment concepts are characterized through the following criteria (Mell and Grance, 2009b; ISACA, 2009; Microsystems, 2009; Craig et al., 2009):

- *Operation*. The usage target group.
- *Management*. The type of authority responsible for the installation, management and maintenance activities.
- *Realization*. Describes whether the Cloud services are deployed within the target organization’s structures (on premise) or externally deployed at a remote facility (off premise).
- *Security Risk*. A low security risk level means that the applications are executed in a controlled local environment, a moderate risk level means that they are executed within a controlled organizational environment and a high risk level means that there is less control of where the applications are executed.
- *Agility*. Describes the level of agility and flexibility to scale applications and utilize Cloud services.

Table 2.1: Cloud deployment models

	Private Cloud	Community Cloud	Public Cloud
Operation	internal	inter-organizational	public
Management	internal, third-party	inter-organizational, third-party	third-party
Realization	on-premise, off-premise	on-premise, off-premise	off-premise
Security Risk	low	moderate	high
Agility	moderate	moderate	high

Based on the *Operational* criterion, *Private Cloud* (PRC) services are visible only to the members of the target organization (enterprise), which have exclusive access to all PRC services, as well as stored and produced data, whereas with *Community*

Cloud (CC), exclusive access to CC services is relaxed to members of the community and with *Public Cloud* (PUC), PUC services can be accessed by everyone.

Private Cloud services are managed by the target organization or subcontracted to a third party. *Community Clouds* are managed by the community or a third party, whereas *Public Clouds* are managed entirely by a third-party, since the third party also provides the PUC services.

Private Cloud services of big enterprises are realized within their computing centers (on premise), where small and medium enterprises achieve higher economic efficiency by acquiring PRC from third party providers (off premise). The same applies to *Community Clouds*, since the crucial factor in determining whether Cloud service should be on or off premise is the size of an enterprise or community. In contrast, *Public Clouds* are realized and provided by third-party Cloud service providers like Amazon Web Services, Salesforce and Google AppEngine.

Executing applications and preserving the data within private computing infrastructures reduces risks for the enterprises by providing greater control of their data. Dedicated – secured and with exclusive access – private Cloud environments are already provided by third parties in the form of *Virtual Private Clouds* (VPCs) (Wood et al., 2009). In the case of VPCs, enterprises have exclusive control of their dedicated environment and contract service level agreements to stipulate how and where the applications and data are to be handled. Within a *Community Cloud*, the applications and data can be processed on computing services, which are owned by the community members. The provisioning and usage policies, as well as service levels of *Community Clouds* are specified within a contract for the target community. The security risks rise with the size of the community. In the case of *Public Clouds*, the applications and data can be handled from any provider around the world or within a specific geographic region. The risk level depends on provider's security policies. Higher security risks can be expected due to the greater number of members, who execute their applications and data on *Public Cloud* services. In contrast to private and community Clouds, service provisioning and usage policies with public cloud services are limited and lack common regularity control (ISACA, 2009).

The level of agility is reflected through the capacities of the Cloud system. In the case of private and community Clouds, applications can be scaled to the available Cloud system resources of the organization or community, whereas with public clouds, the capacity is practically unlimited. Furthermore, utilization of Cloud system resources depends on the needs of the organization and community, which may change over time and produce inefficiencies in how the Cloud system is used in the case of private

and community Clouds. Public clouds, however, can deal with fluctuating demand and supply efficiently.

Hybrid Cloud refers to a composition of the three main Cloud deployment types. A realistic scenario of a *Hybrid Cloud* model is a load-balancing facility in cases of over utilization of private or community Cloud services due to daily or event-driven (project deadlines) computing peaks. In such cases, standardization of Cloud technologies plays a crucial role in enabling the portability of data and applications (applications) (Mell and Grance, 2009b). The characteristics of a *Hybrid Cloud* are aggregated by combining Cloud deployment models. One can imagine that the migration policies of applications from private or community to public Clouds will have to take the security level of the outsourced (for external computing) applications or data into consideration.

2.1.4 Concluding Remarks

The Cloud computing paradigm, i.e., offering computing, platform and application services as utilities, represents the next evolutionary step after *Utility* and *Grid* computing. Cloud computing provides a way to create new software architectures that enable the flexible and efficient provisioning of Cloud services. Furthermore, approaches from autonomic computing (Kephart and Walsh, 2004a) are becoming important in order to reduce and hide system complexity from consumers and help system maintainers monitor preventive and reactive arrangements in order to provide reliable Cloud services.

Realizing an autonomic system with self-optimization capabilities is a complex task that increases with the complexity of the target system, i.e., infrastructure, platform or application services, where each of them may follow different optimization goals involving various technical and economic parameters. The literature in economics, especially (computational) mechanism design, game theory and artificial intelligence uses the concept of *utility functions* (also called *scoring functions* in some cases) to model and optimize decision parameters in order to achieve a well-defined goal (Walsh et al., 2004; Kephart and Walsh, 2004a; Wilkes, 2008). Walsh et al. (2004) illustrated this approach with the management of data centers by translating high-level business objectives (given service level objectives) into lower-level decision steps as part of a utility function. Kephart and Walsh (2004a) presented a framework for designing autonomic systems with policies for action, goals and utility functions. Becker et al. (2008); Wilkes (2008) investigated the use of utility functions to establish service level agreements for computing services between providers and consumers. For example,

the high level business objectives represent the overall preferences of a provider (i.e., to maximize revenue); lower-level decision steps may include increasing the target storage price of a storage unit dynamically, such as when demand and consequently storage utilization increase. A utility function aggregates a set of (weighted) technical and economic parameters into a single value (usually a monetary unit).

In their paper, Foster et al. (2004) discuss the need to make the management of computing systems more intelligent and autonomic. The authors propose the usage of software agents that enable automated and intelligent monitoring of Grid systems, service discovery, application submission and execution tasks. Furthermore, they discuss the need for standardized message protocols, standard APIs and policies to rule the communication of software agents and make it easier to understand their intentions. Another proposal is to automate the selling and purchasing processes through trading agents that negotiate the preferences of consumers and providers in bilateral negotiations or multilateral auctions (Lai, 2005).

The next section discusses the benefits of dynamic price mechanisms for allocating computing services, as well as the need for agents, bidding strategies and message protocols to automate the bidding and matchmaking processes in such markets.

2.2 Why Bidding, Agents and Messages in Markets for Computing Services?

Economic approaches for allocating computing services have been investigated for many years. Many works have focused on the design of market mechanisms for allocating computing services as commodities (see Section 2.3.4). However, there is very little reference to the design of bidding strategies and their efficiency in continuous market mechanisms (i.e., bidders join the market continuously and on demand) with imperfect information in the existing literature. Each of the bidders have different goals and do not share or receive full information about the past, present or future intentions, goals and actions of others. The market mechanism is cleared on a continuous base, as soon as there is a match between available consumer and provider bids in the order book.

The aim of economics is to investigate the mechanisms that enable the efficient allocation of resources with respect to demand and supply. Each resource has a certain value for its owner and different values for the various consumers. Resources are allocated in an economically efficient manner, at a higher or equal price than offered, when they are purchased by the consumers who value them most.

In parallel to the static pricing models for *IaaS* services, *Amazon* started a new pricing model – spot market for EC2 spot instances – where consumers can bid on unused *IaaS* capacity and execute their applications on those EC2 spot instances as long as their bid exceeds the current spot price for the current time frame (on an hourly basis). *Amazon* adjusts the spot price of an EC2 spot instance, based on the current supply (free capacities) and demand (consumer bids) (*Amazon*, 2009). *Offering free capacities of EC2 instances via a spot market incentivizes indecisive consumers to utilize these to execute their low priority and non-time critical applications for lower prices. At the same time, Amazon profits from the resulting economies of scale.*

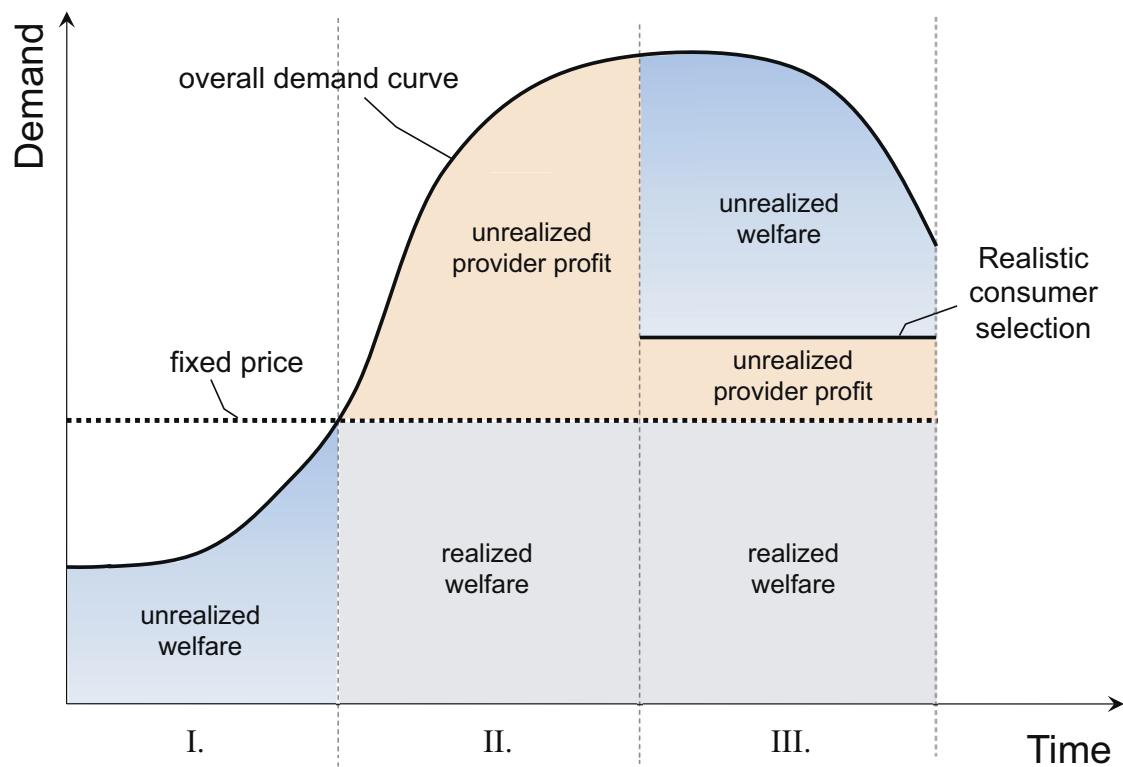


Figure 2.2: Static versus dynamic pricing (adopted from Lai (2005))

A pricing model, in which the target price is established through dynamic bargaining situations like auctions or negotiations based on supply and demand, is called dynamic (or variable) pricing (Lai, 2005). Dynamic pricing is typically used for calculating the price of differentiated and high value products (services). Auctions are established market mechanisms for performing efficient aggregations of fluctuating supply and demand (Wurman, 2001). Figure 2.2 illustrates three different cases,

which describe the relationship of static pricing to fluctuating demand and supply, postulating the fact that market mechanisms with dynamic pricing policies achieve more (economically) efficient allocations of differentiated services than static pricing scenarios:

- *Case I* describes the case where the static (fixed) price exceeds the demand, e.g., the consumer's willingness to pay is lower than the requested price. The welfare (sum of consumer and provider utilities) that would have been gained by selling and using the service is unrealized.
- *Case II* shows a situation in which the demand exceeds the fixed price, i.e., the consumer's willingness to pay is higher than the provider's prices. Consumers are able to purchase the service at the fixed price and a welfare is realized at that price together with the providers. Assuming that the provider can choose the consumer with the highest willingness to pay, the difference between the overall demand curve and requested fixed price is the provider's unrealized profit (Lai, 2005).
- *Case III* is a situation in which there is no common information channel, e.g., globally open order book of all consumer and provider bids for a given trading object (computing service). In this case, the providers do not have full information about the market; they only have information on the visible set of consumer bids in the provider's "local" order book. The consumer bids in the "local" order book have a lower willingness to pay than other interested consumers in the market ("global" order book). Therefore, due to the fact that the price is fixed, the provider supplies the service with unrealized profit. The unrealized welfare refers to the fact that, on the one hand, unaware consumers with higher demand do not get the service, on the other hand, the fixed price of the provider restricts the generation of higher profits.

Traditionally, the price determination process of goods and services is described dynamically through supply and demand curves, which depict the relationship between the prices and quantities (Figure 2.3). The *demand curve*, D , describes the consumer's willingness to pay in relation to the quantity of a service by taking their preferences, endowments and the technology required into account. D is almost decreasing, assuming that consumers will buy more from the service as the price goes down. The *supply curve*, S , represents the quantity and corresponding price of a service that the market can offer when taking the provider preferences, endowments

and the technology provided into consideration. The upward curve reflects the fact that higher prices of a service will stimulate providers to increase the number of provided services in order to increase their revenue (Varian, 2009; Foley, 2010). Gjerstad (2007) states that the time pace of bid submission plays a role in the price finding processes of humans, but also of automated software agents. It has been observed that if the pace of bid submissions of consumers is lower than the pace of bid submissions of providers, then prices are likely to move below the equilibrium price and vice versa. Figure 2.3 depicts this loss of efficiency of consumer and provider surpluses when trades occur out of equilibrium. Here the providers request prices that are lower than the equilibrium price. The dashed lines represent the quantity and price offered for the traded service. In this case, consumer demand to purchase the traded services is higher, consequently, providers will have strong incentive to increase the price and offer more service units.

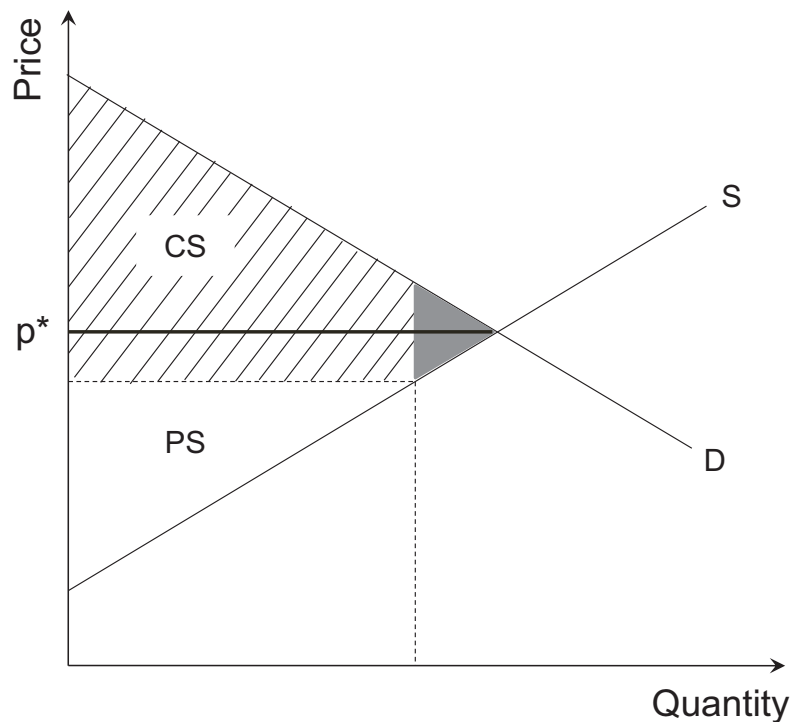


Figure 2.3: Consumers' (CS) and providers' surpluses (PS) out of equilibrium (adopted from Gjerstad (2007))

In general, price finding processes between consumers and providers are described in *equilibrium* and *non-equilibrium* trading theory. A service market is in general

in equilibrium when supply is equal to demand, i.e., in this state, the allocation of the service is most efficient because the amount of the demanded services is exactly the same as the amount of the provided services at the given equilibrium price (p^*). The original idea of price equilibrium was developed by Léon Walras in 1874, who pointed out that such equilibria can be reached through a price adjustment process, also called *tâtonnement* (French for “groping”) or the *Walras tâtonnement* process. In such a scenario, consumer and provider agents shout their price signals for a target good (service). The interactions between agents and the price finding process is coordinated by a central entity called an “auctioneer.” Based on the price signals received, the auctioneer adjusts the price to balance supply and demand by finding an equilibrium price. A trade between agents is permitted only if the trade price is in equilibrium (Cheng and Wellman, 1998).

In non-equilibrium theory, agents follow a *non-tâtonnement* process, where trades between consumers and providers are allowed before the market has reached equilibrium (Schlieper, 1974). The non-equilibrium theory is motivated by the fact that markets are imperfect – online market mechanisms are designed to be computationally tractable and are often executed continuously, services are not homogeneous, traders are not perfectly informed and their supply and demand (functions) changes dynamically over time. Here, the allocation of providers’ services to consumers’ applications takes place in parallel to the price adjustment process. Works showed that *non-tâtonnement* processes converge to competitive equilibria (Negishi, 1962; Chander and Tulkens, 2006; Mukherji, 2008). Market mechanisms such as the *Continuous Double Auction* implement a *non-tâtonnement* allocation processes of supply and demand (Wurman et al., 2001; Gjerstad, 2003). Such market mechanisms are widely applied in practice because of their practicability (see also Section 2.3.5). The trading behavior of consumers and providers of computing services within market mechanisms can be described and analyzed as a *non-tâtonnement* process. Thus, market mechanisms and bidding strategies have to deal with a high degree of complexity due to the reality of imperfect competition, which fosters strategic interactions by traders and leads to inefficient allocation in comparison to equilibrium-based allocation.

The analysis of *tâtonnement processes* assumes perfect competition for homogeneous services, however, computing services are neither storable nor homogeneous, their supply and demand fluctuates and consumers and providers often do not know their preferences or do not want to reveal them truthfully (Rothkopf, 2007). Thus, it requires a comprehensive analysis of (dynamic) allocation processes, in which agents and services are not homogeneous in markets that lack perfect information. These considerations not only play a crucial role in the design of online mechanisms for

market-based scheduling, but also in the design of appropriate bidding strategies. Instantiated into software agents, the bidding strategies can automate bidding processes efficiently and the agents can automatically act on behalf of the consumers and providers (Das et al., 2001; Greenwald et al., 2003; Ku et al., 2005; Kephart and Chess, 2003).

The (software) bidding agents coordinate and execute the bidding processes autonomously, as well as integrate with the heterogeneous system through well-defined interfaces and communication protocols (Kephart and Chess, 2003). A distributed system for market-based scheduling interacts through well-defined (preferably standardized) and common APIs. The system components exchange messages according to a well-defined protocol. The protocol specifies the types of messages, which are exchanged between specific components – application and bidding agent, bidding agent and market and vice versa. Moreover, a semantic specification of message protocols is crucial since there is a need for common understanding, trust, validation and verification capabilities for the messages within the system components (Kephart and Chess, 2003).

Sections 2.3 and 2.4 present the economic and technical environment of a system for market-based scheduling of computing services. Moreover, the sections discuss the challenges that arise when designing and implementing such a system from an economic and technical point of view.

2.3 The Microeconomic Environment

This section defines the economic environment of the target system for market-based scheduling. It starts with an overview of the microeconomic system and the concepts related to it, followed by a definition of the transaction object. Subsequently, the classic economic challenges of mechanism design are discussed and expanded upon with the challenges presented by computational mechanism design theory. The last sections discuss state-of-the-art mechanisms for market-based scheduling, as well as the target market mechanism for this work – the Continuous Double Auction; in the conclusion, a summary of available bidding strategies is given.

2.3.1 The Microeconomic System

According to Smith (1982), a microeconomic system is defined by an *environment* and *institution*. The *environment* \mathcal{E} consists of the following elements:

- $N = (n_1, n_2, \dots)$ is a set of consumer and provider agents
- θ_x is a set of *transaction objects* of type x (like a computing service)
 $x_i = (2\text{ GHz}, 10\text{ GB Memory}, 100\text{ GB Disk}, 100\text{ MB/s})$
- A *utility function* $U = (u_1, u_2, \dots)$ for the N agents
- The set of environment settings, which represent the agent's individual (private) preferences like taste, knowledge or individual skills, are also called commodity and technology endowments (in an experimental environment these endowments are control variables, which are fixed by the experimenter)

The *institution* specifies and executes the market mechanism, which consists of the following elements (Smith, 1982; Wurman et al., 2001):

- The bidding language M , which specifies technical and economic attributes $b_i \in B$. The messages $m \in M$ are exchanged between the market actors – consumers, providers and institution. A message can be a consumer bid, provider offer or market (clearing) message
- The market clearing policy D_x , which determines how the consumer and provider bids are matched
- Policies for the transaction costs and payments D_π
- Policies that indicate the time constraints for starting and stopping the exchange of messages D_T
- Policies for legal rights and usage of the system. Before starting a communication, each agent has to accept the legal rights $D_{contract}$ of the institution
- Policies for information revelation, $D_{bidding}$, indicating which information is public to the participants, e.g., other agents' bids and clearing prices, average prices, etc.

Neumann (2007) presents the overall structure of a microeconomic system (Figure 2.4). The *Transaction Object*, used in this work, is specified in the subsequent section. To interact with the market, the participating agents have to design and implement appropriate bidding strategies, which are influenced by the various policies of the market, the demand of the agents in terms of their preferences, commodity and

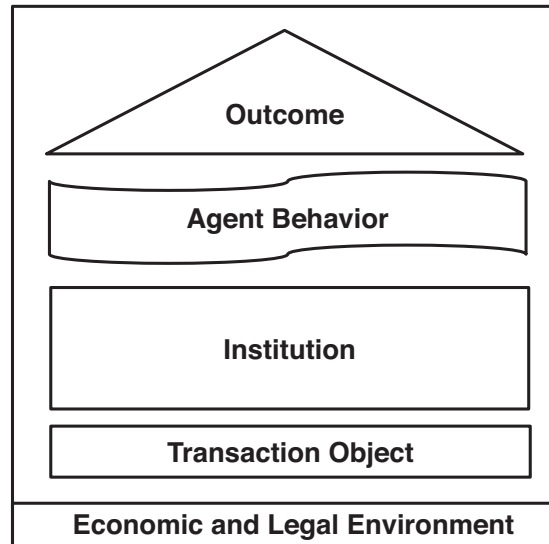


Figure 2.4: Microeconomic system (Neumann, 2007)

technology endowments. The agents' behavior in a microeconomic environment results from their actions and decisions, which are implemented through their bidding strategies.

Agent Behavior has been studied in various contexts in the literature, e.g., in psychology, in economics with respect to mechanism design and game theory, in finance and in computer science, including AI (Smith, 1982; Jennings et al., 1998; Hommes, 2006; Rahwan et al., 2007; van Dinther, 2007). A general approach to modeling agent behavior is the Belief-Desire-Intention model (BDI). *Beliefs* represent the current state of the environment from the agent's perspective. *Desires* refer to the available options (states) an agent may choose to accept. *Intentions* stand for the options (states) selected by the agent. In other words, the agent is continuously reasoning and updating their beliefs about the environment, receiving the available options and deciding which to choose. BDI models are often formalized within logic frameworks. Other works describe the deduction of beliefs from past experience through inference machines. More concretely, in the field of computational economics, consumer agents have to be able to plan their demand – the types and number of applications to be executed, their priorities and estimated completion times. When an application request is received, the consumer agent has to report the application's technical preferences in terms of hardware and software parameters, as well as, priority and related bid (reported maximum willingness to pay). Similarly, provider agents have to be able to

plan their supply – number and type of services, time frames and offers. In the case of market-based scheduling, priority information is indirectly contained within the consumer and provider bids, i.e., the higher the bid, the greater the importance of an application or the quality of a service. Modeling of *Agent Behavior* can be separated into the research blocks *Preference Elicitation and Representation* (De Boer et al., 2001; Sandholm et al., 2006; Chevaleyre et al., 2008), *Design of Bidding Agents & Strategies* (Rahwan et al., 2007; Wellman et al., 2007; Lin and Kraus, 2010) and *Policies for Market Selection* (Vytelingum et al., 2008; Cai et al., 2009; Shi and Jennings, 2010):

- *Preferences Elicitation and Representation.* Preference elicitation is a research field of *mechanism design*, which studies the properties of mechanisms that incentivize consumers and providers to report their preferences truthfully (Sandholm et al., 2006; Chevaleyre et al., 2008). First, however, consumer and provider agents need methods and tools to deduce and *represent* their goals, beliefs, desires and intentions with respect to the conditions (policies) of the market mechanisms. To achieve this, the individual preferences need to be mapped with the conditions (rules) of the market by performing actions and evaluating their outcome. Preferences of consumer applications and provider services is a complex task because these preferences are often unknown and can only be estimated. The literature proposes several techniques to achieve this, starting with reasoning historic actions and their outcomes, statistical methods such as clustering, artificial methods like case-based reasoning, to multi-criteria decision methods like *conjoint analysis*, the *analytic hierarchy process* and *analytical network process* (De Boer et al., 2001).
- *Design of Bidding Agents & Strategies.* After knowing their preferences, consumers and providers need models and tools to govern the consumer and provider bidding processes. The bidding processes are designed into bidding strategies and bidding agents (Rahwan et al., 2007; Wellman et al., 2007). The bidding strategies are software algorithms that implement an action plan of decision rules, where the bidding agents implement the reaction to new consumer or provider requests, as well as to market information – others agents' bids and matching messages. The adaption of the agent's beliefs refers to the fact that consumer and provider bids are executed with well-defined preferences and their outcome is measurable, i.e., it can flow into the automatic adaption of their future intentions when generating new actions. Intelligent strategies will use such information to dynamically adapt to the state of the market and send more

competitive bids, e.g., by applying machine learning techniques (Lin and Kraus, 2010). In the case of markets with perfect information, bidding strategies can apply algorithms to capture opponents' bids and timings in order to extract the patterns of prices, as well as supply and demand to forecast others' behavior. In the case of imperfect markets, agents act under uncertain conditions and apply bidding strategies, which can adapt to incomplete market information and the interaction experience with the environment.

- *Policies for Market Selection.* Another step is selecting the target market mechanism for submitting the consumer and provider bids. The decision of which market to choose depends on the transaction object traded, market policies, transaction costs, allocation efficiency, market dynamics (changing demand and supply over time) and available market information (Vytelingum et al., 2008; Cai et al., 2009; Shi and Jennings, 2010). Furthermore, the state of the markets, availability of computing services and prices are dynamic and bidding agents can gain a profit from choosing the appropriate market and timing based on their past experiences in the available markets. An example of different markets for computing services is the Amazon EC2 spot market, which offers consumers different types of transaction objects – small, medium, large configuration settings for EC2 instances, each traded in a separate spot market.

In the context of Game Theory, the *Outcome* of a system is the set of action-payoff pairs resulting from the strategies taken by agents. The payoffs (also called rewards) are the result of the actions performed by the agents and subsequent allocations at the different stages of the game (Rubinstein, 1985). A mechanism designer can measure the *Outcome* of the system conditionally based on the consumer and provider rewards reported from their performed actions with economic metrics like aggregated consumer utilities, aggregated provider utilities and the sum of both, known also as the welfare of the system. To do this, agents have to reveal their true utility or scoring functions, which is not always the case in real settings (e.g., due to budget constraints or business concerns) (Rothkopf, 2007).

The efficiency of a system is also theoretically described with the *Pareto criterion*, which states that an allocation is *Pareto efficient* if (by a given allocation state) there is no way of making an agent better off without making another worse off. The change in an allocation, where an agent can be made better off without reducing the utilities of others is called *Pareto improvement* (Stavins et al., 2003).

2.3.2 Classification of the Transaction Object

Figure 2.1 in Section 2.1.3 provides a structured view of the differentiated services of the Cloud computing stack. According to their level of abstraction Cloud services are separated into *infrastructure as a service* (low-level computing services), *platform as a service* (for creating and running applications) and *application as a service*.

Goods or services can be separated into *substitutes* and *complements*. Substitute services are those that are interchangeable (e.g., oil and gas; *Tomcat*⁸ and *Jetty*⁹ web service container; free and licensed email clients; etc.) and have a similar value to consumers. Given a set of substitutable services with no or low switching costs, rational consumers are expected to favor the inexpensive substitute. Rising demand of standardized computing services is fostering the provisioning of novel substitutes by reducing proprietary service offerings and the switching costs associated with them. Thus, competition among consumers and providers will increase. Complementary services are valuable to consumers when purchased in a bundle, e.g., CPU and storage, hardware and related proprietary operating system, etc. Complementary services are often proprietary services and important for a business's growth and generating profit. While rising competition through substitutable services may reduce business profits, complementary services can influence profits in a positive or a negative way (Porter et al., 2001).

As specified in Section 2.1.3, the *IaaS* layer represents service types, which are close to the hardware component, i.e., functional units for managing computing, storage and network elements. Infrastructure services are based on standardized hardware connection interfaces and standardized software interfaces for configuration, implementation, monitoring and communication (POSIX¹⁰, SFTP, TLS/SSL to support VPN connections, etc.). Rising supply and demand for *IaaS* services will evolve them into substitutes and increase competition among consumers and providers. Amazon (2009) attempts to increase profits through economies of scale by offering computing services as "spot instances," where the price fluctuates dynamically based on supply and demand. Example computing service instances (CSI) are depicted in Figure 2.1 – small (CSI-S), middle (CSI-M) and large (CSI-L) instances with rising technical properties. Other *IaaS* instances are the storage service instance and *virtual private network*, which can be offered with varying technical properties for storage and

⁸<http://tomcat.apache.org/>

⁹<http://jetty.codehaus.org/>

¹⁰Portable Operating System Interface, <http://standards.ieee.org/regauth/posix>

bandwidth units (e.g., Terra Bytes). Description languages like the *Job Submission Description Language*, *Resource Specification Language* and *Common Information Model* are commonly used in technical descriptions of computational services and communicating technical preferences (Anjomshoaa et al., 2005; Laure et al., 2006; Feller et al., 2007; DMTF, 2010a). Galán et al. (2009) present a promising approach for automated deployment, configuration and execution of applications on top of *IaaS* services. They propose XML schema extensions of the DMTF's *Open Virtualization Format (OVF)* standard for tagging *Key Performance Indicators* for monitoring purposes and flexible network configuration capabilities. The OVF enables packaging application components (application server, databases, operating system, etc.), as well as related environmental parameters into a set of virtual machines. Additional macro (shell) scripts can be used to setup and start the application environment. In contrast to OVF, the OASIS's SDD message protocol provides capabilities to describe the deployment and configuration procedure of software components. The OVF and SDD can be combined in order to achieve a global setup of the application environment in terms of a virtual machine(s), as well as a deployment and configuration plan of the software components within the virtual machines. The *IaaS* platform provides services for runtime and resource management, monitoring and security of CSIs (see Section 2.1.3.1).

Platform services (*PaaS*) enable full management of the application environment through well-defined functionality units like deployment, execution, monitoring, security and billing (Mell and Grance, 2009b). Current *PaaS* offerings are rather proprietary – application owners have to adapt their applications using predefined APIs of the selected *PaaS* provider, which result in higher switching costs when migrating to another *PaaS* provider – and support a selected set of programming languages. Management of the application environment is based on tool as well as APIs (e.g, web services). *Amazon Web Services Management Console* and *CloudWatch* are tools that offer core functionalities for creating, terminating and monitoring (CPU utilization, disk and network I/O) of (EC2) computing instances. Similarly, the *Google App Engine Admin Console* deploying and testing applications, viewing data logs and monitoring the CPU, memory and I/O traffic (Galán et al., 2009; Dunsavage et al., 2010). In contrast to *IaaS*, the deployment of consumer applications on the provider *PaaS* platform is supported by the provider tools. Currently, there is no standard message protocol for platform service descriptions, however, providers could apply the SDD message protocol to provide automated deployment of consumer applications.

The functionalities of *PaaS* services are rather more straightforward than those of application services. Application services, *AaaS*, (Figure 2.1) are implemented in a specific domain context (e.g., email, document or music management). Applications can be implemented as stand-alone services (e.g., “Photo”) or can be combined with other complementary services to achieve a common goal. For example, a “Shop Payment and Inventory Service” can combine many stand-alone services like *Barcode Reader* for transmitting the price information of a good, *Billing Service* to prepare the invoice, *Payment Service* to execute the online payment, as well as *Inventory Service* and *Purchase Service* for warehousing and reordering of scarce goods. Atomic and complex services serve specific domains and differ in their implementation and communication protocols, e.g., SOAP, RSS, ATOM and XML-based communication protocols. Maximilien et al. (2007) proposed an abstract programming model that is able to describe combinations of atomic services. There are still challenges to configure complementary, mostly proprietary, atomic services to complex ones. The challenges arise because of the compatibility of data formats, the semantics of attributes, units and interaction protocols. A major issue of application services is their broad domain of attributes and functionalities, which makes automated match-making and usage a challenging task. The integration of two complementary services is executed by integrators, which write API connectors and semantically map both services to a new integrated service called “mashup”. The automation of these tasks is still a work in progress in the current research (Maximilien et al., 2007; Rosenberg et al., 2008; Weber et al., 2009; Dorn et al., 2009).

Based on this background and current research, *AaaS* and *PaaS* services are assumed to be configured manually (through an integrator) and offered with the *Subscription* or *Pay-per-use* pricing model. The execution, scalability and management of consumer applications on top of *IaaS* services can be automated with existing and standardized protocols. Therefore, *IaaS* services can be rather seen as substitutes and thus efficiently and automatically traded with market mechanisms.

2.3.3 Economic Challenges

Trading computing services is a complex problem. The complexity reflects the design of appropriate and efficient market mechanisms, as well as related bidding strategies. The latter introduces challenges about decisions on how to bid, how to estimate the value of a service, how to predict prices and how to model utility functions. These decisions are often derived from the specified bidding, clearing and information revelation rules of the target market mechanism (Wurman et al., 2001).

Table 2.2: Computational mechanism design desiderata (Myerson and Satterthwaite, 1983; Krishna and Perry, 1998; Shneidman et al., 2005; Parkes, 2008)

Desiderata	Description
<i>Incentive Compatible</i>	This mechanism design property guarantees that agents can only maximize their expected utilities when reporting their true preferences, given that all other agents report honestly.
<i>Individual Rational</i>	Agents get non-negative expected gain by participating in a mechanism, regardless of their valuations.
<i>Allocation Efficient</i>	Given all reported agent preferences, a mechanism guarantees selection of an allocation that maximizes the utilities of all agents.
<i>Budget Balanced</i>	A mechanism is called budget balanced if all of the agent's transfer payments sum up to zero, i.e., payments are not injected into or removed from the mechanism.
<i>Computationally Tractable</i>	A mechanism is computationally tractable if the match-making processes scale well with the number of agents' bids, i.e., real world mechanisms needed to offer a polynomial runtime when matching bids.
<i>Communicationally Tractable</i>	Computational and communicational tractability are closely related because the computational complexity of computing an allocation depends on the number and complexity of the message protocols supported by the target market mechanism (e.g., single bid or combinatorial, centralized or decentralized). Therefore, real world mechanisms have to be designed with a lower complexity of message exchanges and efficient matchmaking capabilities.
<i>Open</i>	Open systems often benefit from innovation and standardization through wider acceptance, contributions and usage by bigger communities compared to many closed systems. Consumers and providers are more likely to trust open source market mechanisms rather than closed source proprietary solutions. The versions of the compiled market mechanisms source code can be easily verified against the deployed version by using checksums.

Table 2.2 presents economic and technical criteria for describing and analyzing market mechanisms implemented in computing systems. The design of a mechanism in terms of policies matters in the sense that it defines the trading and strategy space of the participating agents. The literature of mechanism design has specified economic properties (upper part of Table 2.2), which designers have to think about when designing market mechanisms (Myerson and Satterthwaite, 1983; Krishna and Perry, 1998). The so-called *impossibility result* of Myerson and Satterthwaite (1983) proves that there is no mechanism that can be at the same time *allocation efficient*, *individual rational* and *budget balanced* at the same time. Besides the design of a good market mechanism, online systems (e.g., e-commerce) require the design of scalable, i.e., computationally efficient mechanisms. The discipline, which focuses on both – the design of economic and computationally efficient mechanisms – is called *Computational Mechanism Design (CMD)* (Dash et al., 2003; Parkes, 2008). The CMD properties (lower part of Table 2.2) of a market mechanism increase their practicality in real scenarios through computational and communication tractability, as well as their acceptance and trust through the openness of their realization. CMD implies the application of market engineering (Section 2.3) and software engineering approaches, which postulate the iterative steps of requirements analysis, market design, implementation and tests in both an economic and computational manner.

2.3.4 Market Mechanisms for Scheduling Computing Services

One of the first auction mechanisms for allocating computing services dates back to the 1960s (Sutherland, 1968). In this auction, the consumers bid by putting their bids on a printed time sheet for a selected slot on which they can execute their computations. At the end of the day, the winning consumers are informed of their allocated slots via a recorded telephone message. The microeconomics of computer services (also called “raw computing services,” e.g., CPU cycles, memory and bandwidth) had been discussed in several earlier works (Sutherland, 1968; Nielsen, 1970; Cotton, 1975; Mendelson, 1985), but started to gain attention in the 1990s with the *Spawn* system (Waldspurger et al., 1992) and became a more important area of research with *Utility*, *Grid* and *Cloud* computing (Gagliano et al., 1995; Reeves et al., 2005; Grosu and Das, 2006; Heydenreich et al., 2010; Nassif et al., 2007; AuYoung et al., 2007; Campbell et al., 2009; An et al., 2010). A detailed survey of market mechanisms for computing services is presented by Neumann et al. (2008).

In the *Spawn* system, each provider machine sells time slices through a second price, sealed-bid auction mechanism, also called a *Vickrey auction*. The bids are not visible

to the other agents and the winning agent pays the amount of the second best bid. These types of auctions are called *strategy proof*, i.e, the agents are incentivized to report their true value for the time slice within the bid. A central authority, the *Resource Manager*, manages the auctions of each provider machine. If the application has not finished within the allocated time slice (consumers report their estimated time for their applications), the application is allowed to continue its execution, as long as the agent can pay the current market price of the machine. If the agent does not pay to continue the execution, the application is terminated and the next time slice is offered to the market. The Spawn system does not support check pointing or the migration of applications (Waldspurger et al., 1992). A similar approach uses the *Tycoon* system (Lai et al., 2005), which enhances the allocation schema by introducing an *Auction Share Scheduling* mechanism. Instead of bidding for time slices, the consumer agent bids for machine proportions in terms of CPU cycles. Each application is executed on a virtual machine that runs with the allocated proportion of CPU cycles. The higher the bid, the higher the proportion of CPU cycles received for an application. In contrast to *Spawn*, the *Tycoon* mechanism is not strategy proof (Lai et al., 2005; Feldman et al., 2009).

Grosu and Das (2006) investigated three market mechanisms for auctioning computing services – *first-price auction*, *Vickrey auction* and *double auction*. They discovered that the first-price auction protocol favors providers, the Vickrey auction benefits consumers, while the double auction favors both consumers and providers. Similarly, Regev and Nisan (2000) evaluated the *double auction* and *k-double auction* mechanism for trading CPU cycles in their *POPCORN* system. To buy CPU time for application execution, the applications need to be adapted by using the *POPCORN* paradigm.

Amar et al. (2008) presented a market mechanism, which allows the preemption of “low priority” jobs on a single machine or their migration to other machines. The theoretical results are based on the assumption of zero migration cost, whereas numerical experiments with real world workloads showed that performance of the system is also robust with realistic migration costs. The mechanism was evaluated within the cluster and Grid management system *MOSIX*,¹¹ which provides the functionality for migrating jobs and automatic load balancing among connected clusters (Barak et al., 2005; Amar et al., 2008).

¹¹Multicomputer Operating System for UnIX.

Past and current research mostly focuses on the design of market mechanisms for allocating “raw computing services,” as well as on policies for scheduling computing executed in a *Resource Manager* component. Yeo and Buyya (2006) and MacKie-Mason and Wellman (2006) summarize past and current research of market mechanisms and resource management tools. MacKie-Mason and Wellman (2006) discuss the importance of automating markets and participating agents by well-defined interfaces, as well as by the parameterizing the auction design space in terms of auction, allocation and interaction (bidding) rules (Wurman et al., 2001).

According to the coordination modes, scheduling mechanisms can be categorized into “centralized mechanisms,” where the allocation decision of bids and offers is taken by a central unit, and “decentralized mechanisms,” where the allocation decision is decentralized, i.e., taken by the requesters based on all of the responses received from the environment. According to the allocation modes, scheduling mechanisms can be divided into mechanisms that execute periodically (also called “off-line mechanisms”), and mechanisms that execute continuously (also called “online mechanisms”) (Borodin and El-Yaniv, 1998). Web applications like sensor data processing, price forecasting processing, and video and audio streaming require mechanisms that can allocate computing services on demand in quasi real time (Amazon, 2010b). Therefore, this work focuses on online market mechanisms where the allocation and pricing of computing services is executed continuously.

The aim of this work is to study the design and implementation of competitive and adaptive bidding strategies, as well as bidding tools for trading computing services in Cloud markets. The choice of an appropriate market mechanism was performed based on several considerations and assumptions. Like with the energy market, it is assumed that the number of consumers and providers will be high, have variable preferences, and that supply and demand will fluctuate over time (Shneidman et al., 2005). Furthermore, such a mechanism should be efficient and scale well with a varying number of participants. A well-studied, widely applied and centralized online mechanism is the *Continuous Double Auction* (CDA) (Friedman, 1993; Kant and Grosu, 2005; van Valkenhoef and Verbrugge, 2009; Parsons and Klein, 2009). CDA is a two-sided mechanism, where both consumer and provider bids are permitted (if only bids or offers are permitted, it is called a *one-sided* mechanism). This mechanism is evaluated in the context of a *Public Cloud Scenario* for trading computing services as specified in 2.3.2, but analogously applicable in the context of *Private Clouds*, where the market is internal within an enterprise. In a *Public Cloud* scenario, it is assumed that there are many providers (e.g., computing centers, medium and big enterprises), which have free computing capacities that can be offered to the

public, as well as many consumers (e.g., researchers, small and medium enterprises, and private consumers) which want to consume computing services or cover peak demands. The CDA mechanism is a well-applied, convergent and computationally scalable (tractable) for scenarios with many consumers and providers because it is less message intensive (one message per bid and one for a match; if not matched, the bid expires) as consumers and providers submit their bids to a central authority, which matches the requests, and are therefore, *Computationally-* and *Communicationally Tractable* by design. Furthermore, CDA is *Budget-Balanced* and *Individual Rational* (Parkes et al., 2001). Therefore, the CDA market mechanism seems to be a good candidate for scheduling computing services in the *Public Cloud*.

2.3.5 The Continuous Double Auction

Commonly employed in commodity and financial markets, the Continuous Double Auction (CDA) is one of the most well-studied two-sided market mechanisms (Friedman, 1993; Das et al., 2001; Parsons and Klein, 2009). In this market, consumer and provider bids are matched “continuously” in the sense that the market clears instantaneously on receipt of a bid. If there is no match, the bid is stored in an order book until a match arises or the specified valid time (“time to live”) expires. The matching process in CDA is mostly based on the quantity and price for a target commodity (service), where the price is a representation of the consumer’s or provider’s preferences, i.e., valuation v_j per time unit. Based on a preferred bidding strategy, consumers and providers generate and submit bids (for consumers: $q_j \leq v_j$; for providers: $q_i \geq v_i$) to the CDA market. In the case of consumers, q_j means the maximum willingness to pay; in the case of providers, it means the minimum requested price for a target commodity. A match represents an immediate contract between a provider and a consumer. Thus the consumer is authorized to *immediately consume* the allocated commodity and pay the calculated clearing price π to the provider. The calculation of the clearing price on a match depends on the selected pricing policy. A common pricing schema for the CDA market is *K-pricing* (Satterthwaite and Williams, 1989).

$$\pi = kq_j + (1 - k)q_i \quad (2.1)$$

The choice of the parameter $k \in [0, 1]$ in Equation (2.1) influences the price of the trade. If $k = 0$, the provider sets the price, whereas when $k = 1$, the price is set by the consumer. To have a fair price determination, k is usually set to $k = 0.5$.

In the case of CDAs, applications compete directly and when matched, they are immediately executed on the allocated provider computing instances. Thus, the

maintenance of a waiting queue on each machine is obsolete. For the CDA market, there is no known optimal bidding strategy that the agents can apply. Therefore, the CDA has been widely employed in experimental economic studies using the bidding strategies of different agents to investigate the efficacy of algorithms that automate the bidding processes (Das et al., 2001; Gjerstad, 2003; Vytelingum et al., 2008; Schwartzman and Wellman, 2009).

In a market-based scheduling context, project SORMA, the consumer and provider bids are continuously matched according to a two-phase matchmaking protocol, which is price-based (plus other economic parameters) with integrated technical matchmaking (see Section 5.4) (Borissov et al., 2009b; Nimis et al., 2009, 2008). The CDA performs the price-based matchmaking phase.

2.3.6 Consumer and Provider Bidding Strategies

Auction and strategy selection are closely connected in the sense that a given choice of auction mechanism will affect the choice of target bidding strategy, and vice versa. For example, some bidding strategies perform well in a CDA, but not in other auctions like Dutch or English auctions. This also implies that an agent's success in a particular auction type depends on the selected bidding strategy. Classic approaches to designing (programmable) bidding strategies suggest performing controlled laboratory experiments by letting participants "play" and iteratively develop their own bidding strategies in the defined market mechanism game. Participants develop their own bidding strategies based on the rules of the market, their intuition or experience (Axelrod and Hamilton, 1981). Selten et al. (1997) proposed a strategy development method, in which participants are allowed to play their strategies in a laboratory setting in order to gain experience of the "market game" and then are left to implement their strategies for further investigation.

There has been increased attention on recent research on the automation of negotiation and bidding processes with software agents that apply bidding strategies using information from the environment and that can interact with the market mechanism, as well as against other agents (software or human) efficiently. One of the reasons is that software agents can deal with the complexity of negotiation and bidding in asymmetric environments (different market mechanisms and agent strategies) more efficiently than human agents can, i.e., software agents can extract and aggregate environmental information in near time and store it in databases to be used in future decision-making algorithms. Furthermore, learning algorithms can deal with the uncertainty of preferences and adapt them to maximize a given scoring function.

Another important fact is that human agents suffer from emotional affects¹² and do not follow equilibrium strategies, i.e., “when playing with humans, the theoretical equilibrium strategy is not necessarily the optimal strategy” (Lin and Kraus, 2010). Thus, game theory and computer science became closely connected to elaborate pragmatic solutions in mechanism design, and the design of artificial agents, such as for electronic commerce, monitoring, algorithmic trading in finance markets, negotiation and bidding (Shoham, 2008).

Early approaches to software agents for negotiating and bidding in auctions are investigated in Rosenschein and Zlotkin (1994); Chavez and Maes (1996); Doorenbos et al. (1997); Wurman et al. (1998); Hu and Wellman (1998). Later research explored trading agents and bidding strategies in various fields like financial markets (Das et al., 2001; Sherstov and Stone, 2005; Vytelingum et al., 2008), comparison shopping (called *Shopbot*) agents (Greenwald et al., 1999; Kephart et al., 2000), supply chain management (Pardoe and Stone, 2007) and market-based scheduling of computing services (Wolski et al., 2001; Vulkan, 2003; Li and Yahyapour, 2006; Reeves et al., 2005; Vilajosana et al., 2008). Wellman et al. (2007) give an overview of the various agents and the strategies used in the *trading agent competition*. Phelps (2007) investigated an evolutionary approach for learning the space of bidding strategies. Anthony and Jennings (2003) present an agent framework that structures the design of bidding strategies and introduce bidding algorithms that are able to bid in different market mechanisms like the *English*, *Dutch* and *Vickrey* auctions.

In general, bidding strategies can be classified into *non-adaptive* and *adaptive* strategies. In the *non-adaptive* strategies, the bid generation process does not take current and past market information into consideration (e.g., bids from other participants, clearing prices, market trends and news). Examples of such strategies are the Truth-Telling and Zero Intelligence (ZI) strategies. In the *adaptive* strategies, the generated bid takes the available market information into account. Examples of adaptive bidding algorithms are the *Zero Intelligence Plus (ZIP)* (Cliff and Bruten, 1997), and *Kaplan* and *Gjerstad-Dickhaut* (Gjerstad, 2003) strategies. A comparison of state-of-the-art bidding strategies for the *Continuous Double Auction (CDA)* is evaluated by Das et al. (2001). The authors of the Adaptive-Aggressiveness (AA) strategy (Vytelingum et al., 2008) describe and evaluate a novel bidding strategy for financial markets that implements short and long-term learning behavior that takes market dynamics into account. Simulation results show that the AA strategy outperforms the ZIP and GD strategies for the selected design type of the CDA.

¹²Risk seeking, risk-averse or risk-neutral human agent behavior (Parsons and Klein, 2009).

To automate and optimize agent decisions in bidding processes, designers of bidding strategies often apply machine learning techniques like *supervised*, *reinforcement* and *unsupervised learning* (Weiss, 2000; Müller et al., 2001; Tesauro, 2007). The *supervised learning (SL)* approach is when both input and output data sets exist and are labeled (structured description of the input and output data and dependencies). The SL algorithm performs a classification or regression model of the input-output relationship in order to exhibit and derive a better understanding of system behavior. In the best case scenario, the SL algorithm will find novel correlations based on the labeled input and output data. Deriving a regression model of the system will enable learning to produce correct output data when new input is given. *Unsupervised learning (UL)* refers to the type of algorithms that try to find correlations in a data set that is not labeled, i.e., there is no explicit knowledge other than the raw data. Example of UL techniques are *data mining* (e.g., pattern recognition) and *clustering* of “similar” items based on given characteristics. Another type of machine-learning is the *reinforcement learning (RL)* approach, “which is largely unstudied for systems-management applications” (Tesauro, 2007). The RL approach can be applied in highly dynamic systems with high uncertainty about information and system behavior. An agent using the RL technique can explore a system’s behavior through “trial-and-error interactions,” i.e., by performing various actions and adapting them based on their received payoff (Kaelbling et al., 1996). The general RL approach describes an interactive approach rather than a concrete description of the observed model. It does not say anything about the concrete space of input data, the endogenous relationships, payoff functions or transfer states (generation of a new unknown state or selection of an existing state with the highest commutative payoff). As such, the algorithms that apply RL have to map the “domain-specific initial knowledge” or the model of the environment in a way that enables actions to be generated based on concrete policies and adapted from payoff functions.

A detailed explanation of existing and suitable bidding strategies in the context of this work is presented in Chapter 3, which also includes a description of the *Q-Strategy* (Section 3.4), which is one of the contributions of this thesis.

2.4 The Technical Environment

This section presents the technical requirements for two application scenarios for the market-based purchasing of computing services from external providers. The technical challenges involved in developing an overall system for market-based scheduling is also discussed. These challenges have been derived from the literature, as well as

from the goals of the SORMA project. Its reference architecture, related components, as well as the bidding scenario are presented in the subsequent sections. This work does not aim to evaluate the SORMA system, but rather the developed bidding tools and message protocols within the context of the project, as well as the scenarios presented.

2.4.1 Application Scenarios

2.4.1.1 Batch Supply Chain Data Analysis

The first application scenario represents the class of batch jobs. Batch processing of jobs (applications) is supported by any cluster, Grid or Cloud infrastructure system (Foster et al., 2008). Batch applications consist of all required libraries, input data and configuration files for an atomic execution on any computing system, which fulfills their technical requirements. *TXTDemand* is a supply chain management application, which performs sophisticated analysis and demand forecasting on historic and current product sales (*Sales* and *Replenishment Analysis*) (Windsor et al., 2009). The data from the different consumers is provided as input to the *TXTDemand* batch applications. The batch applications are executed during the night, each of them with a duration of between one and ten hours. The results of the *TXTDemand* analysis are used by the related enterprises to make daily decisions regarding their sales and replenishment strategies.

The consumers provide their sales data to the *TXTDemand* provider in a well-defined format. The different *TXTDemand* batch jobs are then executed on the provider's local infrastructure or on externally purchased computing services. The *Bid Generator* tool executes the bidding processes to purchase the required computing service configurations from the *Computing Service Market*. The result of a bidding process is the allocation with the established service level agreement from the matched technical and economic attributes of the consumer and provider.

The *TXTDemand* scenario defines the following requirements for its integration in the SORMA system for market-based scheduling (Windsor et al., 2009):

1. A system for market-based scheduling has to provide the possibility of negotiating service level agreements and instantiating related contracts. The negotiation process should include technical parameters for raw resources like CPU, memory, storage and bandwidth. A service level agreement has to include information about the duration of a job and the maximum reservation time for the computing service provided.

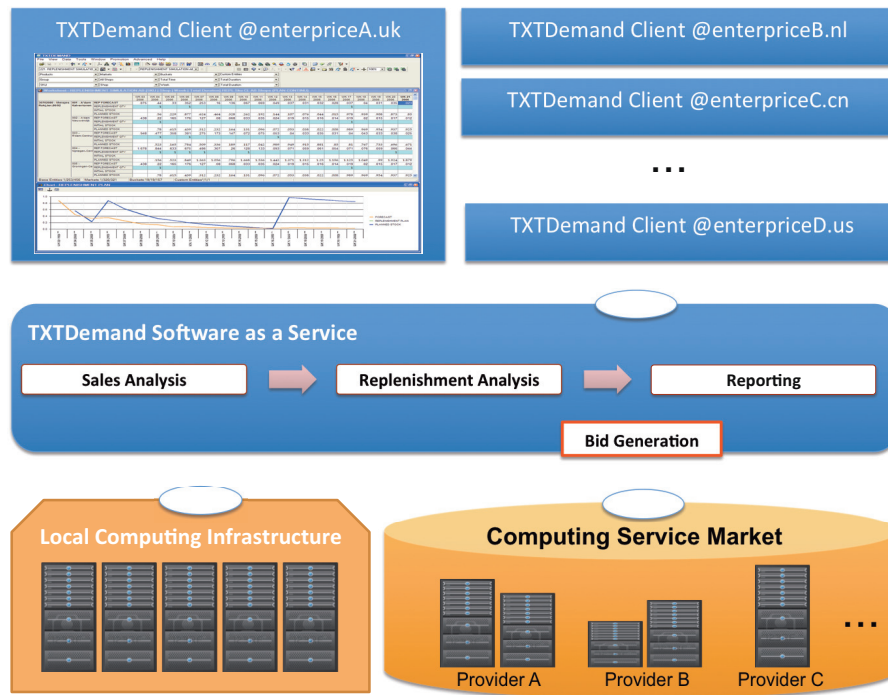


Figure 2.5: Scenario for supply chain data analysis with local and externally purchased computing services (own representation, based on the SORMA project) (Windsor et al., 2009; Nimis et al., 2008)

2. The negotiation protocol has to consider situations in which resource providers are forced to break certain service level agreements because they cannot cope with them. The economic impact of such events and their causes have to be evaluated and appropriate mechanisms have to be developed to address them (e.g., compensation payments, which are also called penalties).
3. To increase acceptance and trust in the system, providers of computing services have to be evaluated according to well-defined indicators like reliability, performance, etc. Furthermore, privacy, data protection and security has to be ensured with well-defined policies that are enforced and easy to demonstrate since the *TXTDemand* application handles private and sensitive customer sales data.
4. The market-based scheduling system has to support different payment models like pay-per-use, dynamic pricing, prepayments and post-payments.

5. Flexibility has to be ensured in case consumers need more computing services than initially agreed.
6. The bidding tool, *Bid Generator*, may be used by non-experts, so the API and UI should be clear and simple enough to these kinds of adopters. The *Bid Generator* should support consumers and providers in the preparation of bids, and help them understand the economic impact of their actions.
7. The transfer of consumer data and the bidding processes need to be fast and performed over a secured communication line, which is part of the system's communication protocol.

These requirements are used as a basis for the developed and realized models of this work.

2.4.1.2 Interactive Sensor Data Analysis

The second scenario represents the class of interactive Web service-based applications. *Visage* is a system for advanced video data analysis based on sensor data such as from cameras (Figure 2.6). When a motion is detected, the camera (client) starts transmitting the video data to the *Visage* system. The *Motion Detection* component identifies the frames in which the activities are detected. The *Object Recognition* component performs a pattern recognition analysis by identifying the moving objects between a sequence of frames. The process finishes with the creation of a report, which is sent back to the requesting client in terms of video and description data of the objects identified (e.g., picture of a moving person and a car, Figure 2.6). Such a system implements an online and on-demand software service for sensor-driven video data analysis that can be applied to automate security issues in and around buildings. As a software as a service approach, *Visage* offers a Web service interface, in which camera sensors distributed around the world and in different enterprises can connect and evaluate their video data. The allocation of a client to a *Visage* node is performed by the *Visage Service* according to the reported consumer technical and economic preferences. When an allocation is received, the client component (camera application) starts submitting the video data.

The service provider maintains its own local computing infrastructure of cluster machines to run the *Visage* system. The demand for computation power depends on the number of consumers and cameras that they have installed. In such a scenario, the ratio of average demand to high demand depends on time and geographic factors, as

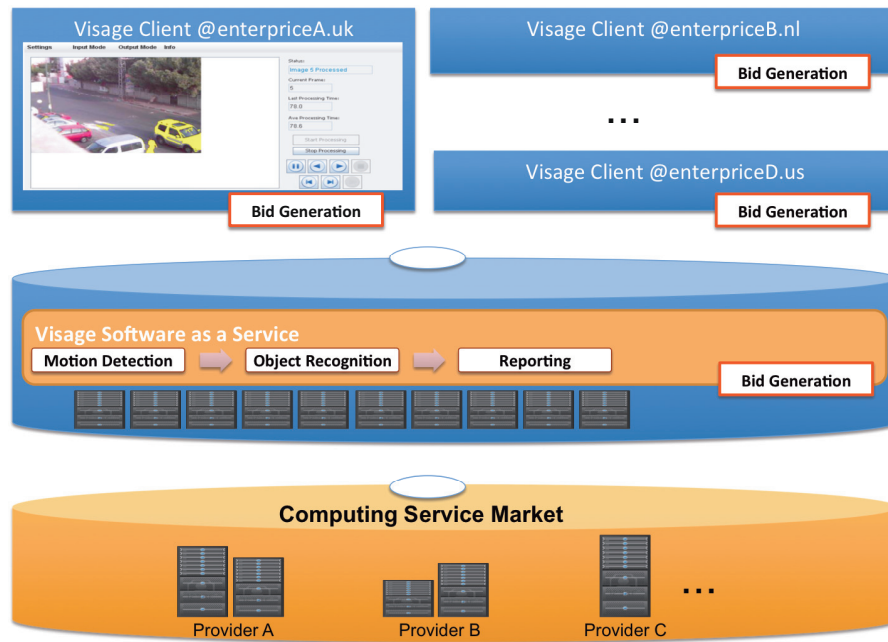


Figure 2.6: Scenario for on-demand video data analysis with local and externally purchased computing services (own representation, based on the SORMA project) (Windsor et al., 2009; Nimis et al., 2008)

well as on consumer preferences (Windsor et al., 2009). The demand for additional computing services (i.e., in addition to the locally existing services) is calculated by the “PeakDemand” component, which is part of the *Visage* system. That component purchases the required computing services from the *Computing Service Market* on demand by invoking the *Bid Generator* component. Therefore, the *Visage* system scales on demand based on the number of connected sensors and calculated demand on the “PeakDemand” component.

In order to realize the described scenario, the following requirements have to be fulfilled by the SORMA system (Windsor et al., 2009):

1. On-demand deployment of the *Visage* system on external computing infrastructures.
2. Automated purchasing and allocation of external computing services from the market according to well-defined and implemented bidding strategies.
3. The *Visage* system should be deployed on as many computing services as needed. Inefficient utilization of computing services (waste) should be prevented

through maintenance of a limited local computing service infrastructure. Peak demand is covered by external providers on the market.

4. *Visage* clients may request several resources simultaneously and the allocation has to take all simultaneous requests into account.
5. The *Visage* system and clients communicate through secured interfaces and all data is protected from unauthorized access.
6. The *Visage* system defines specific requirements for computing services, which are to be expressed in the generated bids and considered in the matchmaking process. Furthermore, the result of the market-based allocation is a service level agreement, which has to be fulfilled by the external provider.

This work focuses on the trading of computing services and the related communication protocols. In this context, only the relevant parts of the *Visage* scenario will be considered – purchasing computing services through the *Computing Service Market* and defining the required bidding language. The definition of (software) application specific attributes is not part of this work.

2.4.2 Technical Challenges for the Market System

Systems for market-based scheduling of computing services can be compared to current e-commerce systems like eBay, Amazon Web Services and Google AppEngine. They offer interfaces that allow the automation of purchasing and offering processes for goods and online services. Purchasing and offering computing services with market mechanisms are complex tasks. Such processes require less human intervention and can be automated by using intelligent software agents in order to achieve higher efficiency in decision making processes (Kephart and Chess, 2003; Cheliotis et al., 2005). Software agents can monitor and manage many more computing services and applications simultaneously than human agents can. Furthermore, software agents can process huge amounts of data in quasi real time (Foster et al., 2004; Wellman et al., 2007; Feigenbaum et al., 2009).

Following is a list of the technical challenges that need to be addressed when designing the components of a system for market-based scheduling.

Automated Bidding. Consumers and providers require configurable bidding agents, which automate the processes for trading, provisioning and usage. In a realistic setting, this involves designing bidding strategies that can trade in different auction

types with heterogeneous and homogeneous agents, as well as with bounded information about the actions of other agents in the market (Parsons and Klein, 2009).

Bidding Language. To express their technical and economic preferences, consumers and providers need a well-defined, compact and concise term language. In the literature, bidding languages for auctions that trade single and combinatorial goods (services) are analyzed. Matching multiple attributes and weights is an optimization problem, which is NP-Hard.¹³ In order to specify their preferences, consumers need to make decisions regarding the multiple attributes of a computing service configurations for their applications. Similarly, providers have to decide, which computing service configurations are likely to be demanded. Methods for preference elicitation and statistical prediction of market information are used to derive decisions about the supplied and demanded computing service configurations (van Ittersum et al., 2007; Sandholm and Lai, 2007; Kiekintveld et al., 2009).

Automated SLA Creation & Enforcement. As a result of the matching process, a binding and legal contract between a consumer and a provider is created. Both providers and consumers need guarantees in the matched terms in order to be incentivized to offer high-quality services, as well as to execute applications in the Cloud without concerns about security, trust or the quality of services (Wilkes, 2008; Becker et al., 2008).

Trading Platform. Trading platforms should be able to run multiple (online) market mechanisms in a scalable and secure way, as well as communicate and match multiple bids and offers efficiently. The TAC¹⁴ and CAT¹⁵ tournaments encourage research in the design of market mechanisms and bidding strategies that are able to automate bidding and matching processes in a variety of auction mechanisms and commodities in changing environmental conditions (Cai et al., 2009).

Security & Trust. Security and trust in *IaaS* should be provided through transparent auditing, fair matching and enforcement of SLAs. Authentication for consumers and providers is done through certificates and they communicate using standardized security protocols and mechanisms. All messages submitted to the market must be signed and validated before they are considered legal and binding (Nimis et al., 2008, 2009).

¹³Non-deterministic polynomial-time hard (Parsons and Klein, 2009).

¹⁴Trading Agent Competition: Focus on agent design strategies.

¹⁵CAT is the reverse of TAC and comes from *CATallactics*, the science of exchanges: Focus on market design (Cai et al., 2009).

Auditing Services. In order to support the billing processes and reproduce the system outcome (e.g., *fault-tolerance* property), market messages (e.g., bids, offers and matches) have to be logged and timestamped (Nimis et al., 2009).

Market Directory & Information Services. Information about available auction mechanisms and traded computing services has to be stored in registries (“green pages”) and made findable through a query language, according to the technical specification sought. Related market information services have to provide consumers and providers with aggregated information of current and past prices, as well as the type and number of supplied and demanded services (Brunner et al., 2008).

Resource Management. Provider’s resource managers perform the actual allocation of the applications received to their computing infrastructures based on the match-message received from the market. Therefore, providers design their own scheduling policies for the allocated consumer applications with the aim of achieving economic efficiency and consumer satisfaction (Macías et al., 2008). Moreover, providers facilitate consumers’ applications with the requested computing service descriptions according to the service level agreement document (the SLOs are part of the match-message), as well as to the specified payment and penalty conditions (Borissov et al., 2009b; Becker et al., 2008).

Infrastructure Service Engineering. Providers of infrastructure services have to apply the APIs that are for executing consumer applications and offer their free capacities to the market. Internally, the profitability (higher margins of market prices and operation costs) of their computing infrastructure not only depends on the economic management of service level agreements, but also on the continuous improvement of their hardware resources in order to reduce energy costs and gas emissions (Berl et al., 2010; Brown and Reams, 2010). Projects like Facebook’s Open Compute Project¹⁶ work on energy efficient hardware technologies that reduce overall energy consumption for computation, storage and communication tasks. Moreover, like the Top500 project, the Green500¹⁷ provides rankings of the most energy-efficient supercomputers in the world.

Cloud Application Engineering. In order to utilize *IaaS* services, application developers have to apply the APIs and tools of the provider. In a market-based scheduling scenario, the definition and application of common standards and tools is crucial for

¹⁶<http://opencompute.org>

¹⁷<http://www.green500.org>

the practicability trust and acceptance of such a system. Open standards from organizations like the *Open Grid Forum*, *Distributed Management Task Force* (DMTF, 2010a), the *Open Science Grid* and the *Open Cloud Manifesto* offer transparent adoption of *IaaS* services and reduce lock-in effects for consumers (Nelson, 2009). The engineering of Cloud applications is not an aim of this work, however, it is an important area for future research and its acceptance in computing service markets.

Licensing. Software licensing is often restricted to local deployment scenarios. Licensing models for commercial software has to be made compatible for a Cloud-based application execution (Armbrust et al., 2009). An open issue of Cloud service provisioning and usage is the transfer of licenses between applications and third-party software, which is not part of this work, however, it is addressed in the SmartLM project (Cacciari et al., 2010).

Technically, this work focuses on the design, development and realization of tools for automated bidding and related message protocols in Chapters 4 and 5. The next section presents the reference architecture of the SORMA project and the technical challenges associated with it as the basis for the contributions of this thesis.

2.4.3 A Technical Architecture for Market-Based Scheduling

The design of market mechanisms and bidding strategies is essential in achieving a system for the market-based allocation of computing services. There are challenges in realizing such a system with respect to the clear separation of economic and technical concepts in its architecture and components.

The results of this work have been explored and elaborated in the context of the SORMA EU-Project¹⁸ (Self-Organizing ICT Resource Management). Figure 2.7 depicts a simplified version of its logical architecture and the components contained therein. The detailed structure of the components is broken down into four logical layers. The SORMA architecture is a technical view of a microeconomic system, where the *Open Market Middleware* (Layers 1 and 2) represents the institution running the market mechanism (e.g., *Trading Management* executes the CDA mechanism) and defining the legal and communication rules (e.g., *Contract* and *SLA Enforcement and Billing*), Layer 3 represents the agents and their strategies and Layer 4 represents the transaction objects, i.e., computing services for *IaaS* applications. The following sections give a brief overview of the SORMA layers and their components.

¹⁸www.sorma-project.eu

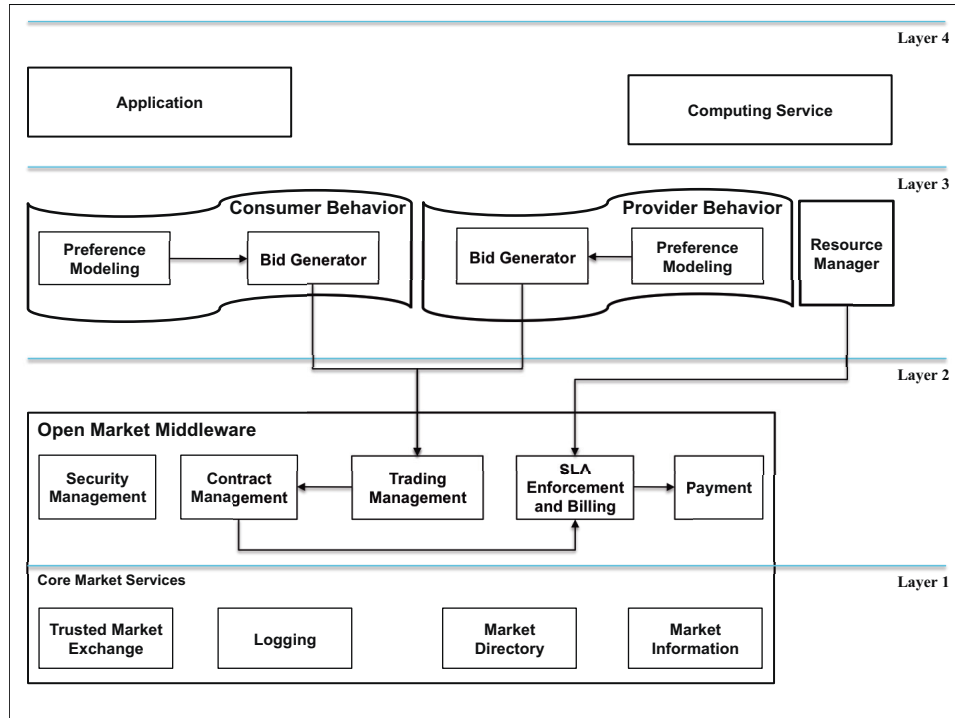


Figure 2.7: SORMA architecture

2.4.3.1 Layer 1: Core Market Services

Layer 1 provides the core infrastructure services of the *Open Market Middleware* (Chacin et al., 2008).

The *Trusted Market Exchange* provides an infrastructure for message exchange among the components of the middleware, as well as the related parties that are part of the auctioning process. It assures that the related messages are routed to the addressed party in a secure and reliable way. This service allows asynchronous and state-free communication among the connected middleware components.

The *Logging* service keeps a registry of the transactions executed on the market for auditing purposes, e.g., for *Contract Management*. Furthermore, it maintains the state of the active messages and thus allows renewal when a party is reconnecting (*Fault-tolerance*).

The *Market directory* is a registry of available auctions for different computing service

types (e.g., CSI-S-Auction, CSI-M-Auction, CSI-L-Auction, etc.). These services can be compared to a UDDI's "green pages," where each commodity (e.g., CSI-S) is associated with a technical description, as well as the endpoint reference of the target auction (e.g., CSI-S-Auction). It provides the functionality to discover the computing service commodities offered based on the technical preferences.

The *Market Information Service* provides aggregated price information and historical statistics of market indicators (e.g., maximum, minimum or average price of a given time interval) for selected commodities. Agents can subscribe to the service and perform queries through the interfaces provided (Brunner et al., 2008). Bidding strategies can use this information to optimize the timing and value of their bids with respect to the type of application.

2.4.3.2 Layer 2: Open Market Middleware

Layer 2 contains the components, which execute the market mechanisms where bids and offers are matched (Nimis et al., 2009, 2008). SLAs are the result of the match-making processes. The execution of an SLA is monitored by the *SLA Enforcement and Billing* component.

The *Trading Management* component is a framework for developing and running market mechanisms (Sections 2.3.5). Market mechanisms implement the rules of how bids and offers are matched, technically and economically, and the rules of when and what to communicate within the exchanged messages (communication protocol). The trustworthiness of the messages is certified by a *Certificate Authority* and monitored and validated by the *Security Management* component.

Contract Management (CM) receives the result of the matchmaking process and transforms the corresponding pairs of bids into mutually agreed contracts. As part of the matchmaking process, the technical and economic preferences are transformed into a *Service Level Agreement (SLA)* document, which defines the *Service Level Objectives (SLOs)*, also called *Key Performance Indicators* of the agreement. The CM component serves as a repository for contracts and initiates the enforcement process for the SLAs.

The *SLA Enforcement and Billing* component receives, monitors and enforces the contracts. It requests information for each SLA periodically and keeps track of the specified SLOs. Based on the current states of SLOs of the applications, the resource management component is responsible for the allocation of consumer applications in a way that maximizes the utilities of the provider and consumer. If the provider fails

to meet some or all of the agreed SLOs, the bill is discounted by a penalty payment (Becker et al., 2008). Finally, when the contract is finished, the final payment is calculated and executed through the *Payment Component*.

The *Payment* service provides a unified interface to an online payment service, which supports recurring micropayments.

Security Management provides the agents with single sign-on entry to the market. Consumers and providers register to a Certification Authority to get a valid certificate in order to sign the submitted messages. The market is executed by a trustworthy institution, and the components within the middleware are running behind a secured infrastructure. Only messages from certified agents are accepted and routed between the middleware components. In SORMA, the authorization and signature of bids was realized with the Security Assertion Markup Language and related tools (Nimis et al., 2009; Armando et al., 2008).

2.4.3.3 Layer 3: Consumer and Provider Bidding Tools

This layer contains components, which assist consumers and providers in automating the bid generation processes.

The *Preference Modeling* component provides the format of the message protocols that consumers and providers use to express their technical and economic preferences through the transaction object (Borissov et al., 2009b). It is assumed that they can estimate their preferences and value bounds by applying the methods of the *Preference Elicitation* theory (see Section 2.3). A part of the economic preference is the selected bidding strategy, which applies the implemented decision rules to generate bids and offers. The bidding strategies are implemented and executed within the *Bid Generator* component.

Bid Generator is the component that connects consumers and providers to the market and trade on their behalf, based on their preferences and selected bidding strategies (Borissov and Wirström, 2008; Borissov, 2009; Borissov et al., 2010). To do this, *Bid Generator* is represented through an agent framework, which offers well-defined interfaces for the API units – *Learner*, *Bidding Strategy*, *Market Connection* and *Security*. Based on these API units, consumers and providers can already use the implemented bidding strategies and policies or develop and test new ones.

The *Resource Manager (RM)* is a provider component that serves the management of the provider's local resources (e.g., machines, computing instances, storage, etc.). The resource manager is designed for direct negotiations and for auctioning through the

Bid Generator component. To perform direct negotiations with consumers, the RM utilizes local policies for resource allocation and monitoring, application execution, SLA enforcement, as well as price discrimination through client classification. To offer resources on the SORMA market, the RM connects to the *Bid Generator* component by reporting the description and reservation price of the computing services offered (Püschel et al., 2007; Macías et al., 2008). The allocated applications are executed according to the market match, i.e., the SLA. Furthermore, the RM can compensate for higher local resource demands by buying computing services from the market.

2.4.3.4 Layer 4: Applications and Resources

Layer 4 refers to the type of transaction objects, which are tradable with the SORMA system (Section 2.3.2) – computing service instances.

On the provider side, a business engineer specifies the type and configuration of a *computing service* to be offered on the market according to the provider business plan. To specify the technical and economic terms of the bid, the provider applies the tools and message protocol from the *Preference Modeling* component. The computing services are managed within the *Resource Manager* and offered on the market by the *Bid Generator* component.

To execute their applications in the Cloud, consumers first have to prepare their applications so that they can be easily transferred and executed on the allocated computing service instance. Consumers use the same tools as providers – *Preference Modeling* to initialize their technical and economic preferences for the application and *Bid Generator* to execute the bidding processes.

2.4.4 Bidding Scenario for Computing Services

Figure 2.8 depicts a general overview of the *Bid Generator* integration in SORMA’s market-based scheduling environment. The environment consists of three actor types – *Consumers*, *Providers* and *Institutions*. Each consumer has applications (*apps*) to execute, which are queued, executed, monitored and terminated by the *Application Orchestrator* component. The *Application Orchestrator* component is a consumer component that manages the deployment and execution of consumer applications (Section 2.4.1) on local and external computing services. In the case of external computing services, the *Application Orchestrator* invokes the *BidGenerator* by submitting the “request for bid,” \mathcal{P}_j . \mathcal{P}_j contains well-defined or estimated technical requirements and valuations for the application j , which are reported to the *BidGen-*

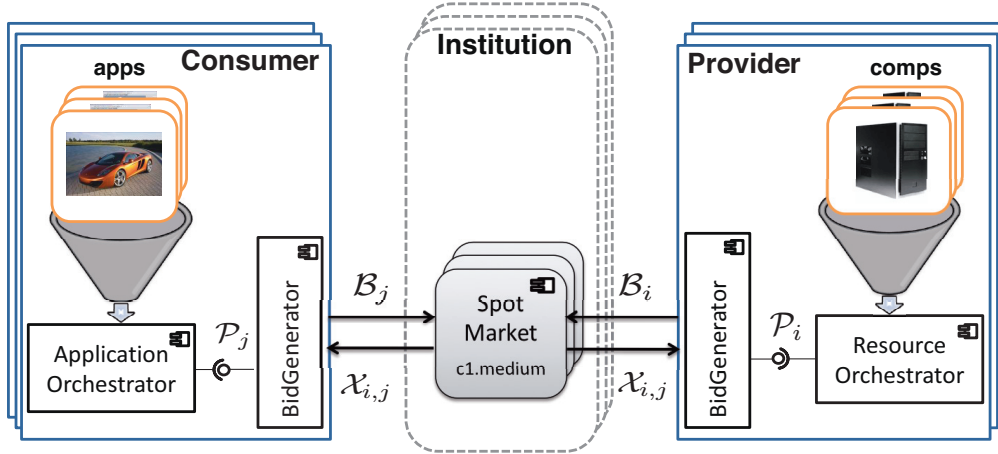


Figure 2.8: Scenario for bidding on computing services

erator component (a description for \mathcal{P}_j is provided in Section 5.3.1). The technical requirements and objectives are manually initiated by the application owner and can be automatically estimated using a regression analysis of the historic data of the same application type and a similar amount of input data (Ali et al., 2004). The *BidGenerator* component provides methods and realizations of state-of-the-art and novel bidding strategies (Chapter 3) and bidding agents (Chapter 4). *BidGenerator* interacts with the environment according to a well-defined communication protocol (Chapter 5).

Providers of computing services (*comps*) maintain their own infrastructure and offer free capacities on the market. Similar to the consumer case, the provider computing infrastructure is managed by a so-called *Resource Orchestrator*. The *Resource Orchestrator* creates, prepares, monitors and terminates (virtualized) computing services according to the provider's local scheduling and business policies (Macías et al., 2008; Püschel et al., 2007; Wilkes, 2008). For each of the free computing service instances, the *Resource Orchestrator* sends a \mathcal{P}_i (Section 5.3.1) to the *Bid Generator* component with a detailed description of their technical descriptions, reserve prices and selected bidding strategy implementations (Sections 3.3.4 and 3.4).

According to the selected market and bidding strategy, the consumer's and provider's *Bid Generators* generate bids and submit them to the market in the form of \mathcal{B}_j and \mathcal{B}_i messages (see Section 5.3.3 for details). The market mechanism performs a technical and economic matchmaking (Section 5.4) of the consumer and provider bids and creates a match-message, $\mathcal{X}_{i,j}$ (Section 5.3.4), which is sent back to the matched

consumer j and provider i . The example on Figure 2.8 depicts a spot market for a computing instance called *c1.medium* with a well-defined specification of technical attributes, similar to the Amazon EC2 offering (Amazon, 2010a; TheCloudMarket, 2010). The market mechanisms are implemented and executed within the market platform called *Trading Management*, which was developed, implemented and tested within the SORMA project (Nimis et al., 2009).

2.5 Research Methods

Bidding strategies implement decision rules according to the information available and state of the environment. The analysis of such systems becomes complex when supply and demand, the number of participants and their applied bidding strategies vary over time. Game theory offers tools to express agent decisions and their relationships for simplified games usually with few competing agents and an idealistic set of assumptions about the environment’s rules and information. Game theoretic models, however, quickly become unfeasible as the complexity of the system grows, which is a reflection of the “strategies space, number of agents, degree of incomplete and imperfect information, and dynamism.” (MacKie-Mason and Wellman, 2006). Laboratory experiments are designed to study the behavior and decision making of humans under certain conditions and experimental settings. In a laboratory setting with human participants, the setting of a mechanism and strategy space is biased and limited by the competencies, dependence on and sentiments of non-expert human participants. It is not only the normative science of an economic system that matters, but also the context and implementation details of the computing service markets and agents. An example of a complex economic system are the FCC¹⁹ spectrum auctions. Even though it was designed by some of the best auction theory and agent strategy researchers, given the rules of the market, an analytical solution of the game has not been proposed (MacKie-Mason and Wellman, 2006; Wellman, 2006). The design of mechanisms and strategies for computing services has a comparable complexity; thus there is less potential for analytical solutions to be proposed in realistic settings.

This work aims to investigate the application of software agents and analyze the outcome of sophisticated bidding strategies that automate bidding processes in a distributed system for market-based scheduling. Such complex analysis is performed

¹⁹Federal Communications Commission.

with methodologies proposed by Tesfatsion (2006) – *Agent-based Computational Economics* (ACE) – and by MacKie-Mason and Wellman (2006) – *Empirical Game-Theoretic Analysis* (EGTA). ACE studies economic processes, which are “modeled as dynamic and distributed systems of interacting agents [...]” as “[...] part of a computationally constructed world” (Tsfatsion, 2006). The application of ACE is “an attempt to achieve a more transparent and realistic representation of real-world systems involving multiple distributed entities with limited information and computational capabilities” (Tsfatsion, 2006). The EGTA of MacKie-Mason and Wellman (2006) is a supplement to the ACE methodology that proposes systematic evaluation of market mechanisms and trading strategies in computational experiments. This approach postulates the construction of an agent-based world that models the key assumptions and aspects of the economy investigated. This work adopts these two approaches for the field of market-based scheduling in the following way:

1. *Mechanism Design*. The selected mechanism for market-based scheduling of computing services is the *Continuous Double Auction*. Its *Computational Mechanism Design* properties – *Budget-Balanced*, *Individual Rational*, *Computationally Tractable* and *Communicationally Tractable* – makes it well-suited for on-line settings and the selected scenario. The *bidding rules* define the continuous matching of arbitrary orders of consumer and provider bids without any time restrictions or time segmentations. The matchmaking process is performed in two phases – a price-based (plus additional economic parameters) phase and a technical matchmaking phase. The market clearing rules are applied as soon as a consumer and provider bids matches in terms of economic and technical parameters, where the clearing price is calculated with the *k-pricing rule* (Section 2.3.5).
2. *Bidding Strategy Design*. The market rules define the scope of the strategy space. Bidding strategies are designed according to the parameter space of the market mechanisms. Such strategies can implement simple decisions like random bid generation or sophisticated decisions by utilizing available market information and historic data. The bidding strategies are the logical part, which implement the decision making steps according to the signals received from the market or other agents. The bidding strategies are instantiated into agents, which interact with the environment through the exchange of messages – e.g., agent’s intentions (bids) and market messages (matches, and market information like the bids and matches of other agents). The design desiderata

of bidding strategies, an investigation of existing bidding strategies and the presentation of the *Q-Strategy* is part of Chapter 3.

3. *Agent Design.* Consumers and providers require tools to implement agents and bidding strategies in order to interact autonomously with the market. A flexible framework for automated bidding aims to provide a methodology for implementing agents and strategies in an effective and straightforward way. Moreover, the framework should facilitate the simple and methodological evaluation of agents and strategies in various settings. Chapter 4 focuses on the definition of bidding agents and their properties, the elaboration of existing agent frameworks and agent design methodologies, as well as the presentation of the BidGenerator framework.
4. *Message Protocols Design.* The definition and adoption of common protocols for exchanging consumer requirements, provider offers, bids, matches and market information is a crucial part of a system for market-based scheduling. Developers of agents and bidding strategies need to understand the semantics, type of messages, and the market information in order to design and implement successful bidding strategies according to their needs and design requirements. Chapter 5 focuses on design desiderata for message protocols and presents the specification of novel message protocols for market-based scheduling.
5. *Experiment Design.* To analyze the economic system, one needs to define the settings of the market, the number of agents and their selected bidding strategies. The strategy profiles of the agents can be homogeneous or heterogeneous – all agents utilize the same bidding strategy or each agent applies a different one. Another part of the experimental designs is the generation of artificial and real-world input data. Historic real-world data for cluster job profiles can be taken from web archives (Feitelson, 2010), where missing data like the private valuations of consumers and providers can be artificially generated only with well-applied statistical distributions like normal and uniform distributions (Brooks et al., 2003; Sandholm et al., 2008). Section 6.1 presents the overall design of the agent-based experiments and Section 7.1 presents the methodology for the technical and performance analysis.
6. *Outcome Analysis.* Successful (matched) bids for a computing services result in an allocable and measurable outcome is called a reward. The overall outcome of an experiment setting is measured according to the reported consumer and provider rewards for each of the allocations in this setting. The reward

(scoring) function for consumers and providers can be different and depends on their private goals like minimizing makespan and payments or maximizing profit (Heydenreich et al., 2010; Parsons and Klein, 2009). Concretely, the overall outcome of each of the experiment settings is measured with *aggregated consumer scores*, and *aggregated provider scores* metrics, and the sum of both, also known as the *welfare* of the system. Section 6.2 presents the results and analysis of the results according to the specified metrics.

Market and strategy design are challenging processes, which are dependent on each other in the sense that the definition of market rules affects the selection and design of bidding strategies, which will affect the outcome of the agents applying them. The presented methodology aims to develop the theory and tools to perform realistic experimental scenarios for bidding strategies in highly dynamic online (imperfect) markets.

Part II

Design and Implementation

Chapter 3

Economic Design of Bidding Strategies for Market-Based Scheduling

THE landscape of today's software services is highly heterogeneous. Consumer applications are realized in different programming languages and run on different hardware and operating systems with respect to their usage scenarios and requirements (e-mail, banking, social apps, office apps, statistical apps, data analysis apps and many other services). Providers maintain differentiated computing infrastructures (HPC computing centers, campus computing centers, and computing centers of large, medium and small enterprises), which are composed of hardware from different manufacturers, which is updated in different time periods. Therefore, in a market-based scheduling scenario, consumers will have a different demand for computing services for their applications and providers will aim for different business models when offering their free capacities on the market. The heterogeneities of consumers and providers will affect their decisions and behavior when bidding in markets for computing services. Such decisions include the time of market entry, the time of market exit, the number of submitted bids and how they are generated. Bapna et al. (2004) analyzed the bidding strategies of consumers that buy goods on online markets such as *eBay*. They identified five types of bidding strategies, in which efficiency (profit) was evaluated with observed real data over two consecutive years. The results showed that consumers learned to improve their bidding strategies in the second year. An important observation was that consumers, who used software bidding agents, achieved the highest profits in comparison to other bidders.

The design and implementation of bidding strategies for specific auction mechanisms automate bidding processes, but the algorithmic decisions implemented influence the outcomes of the agent. In the literature, various bidding strategies are proposed, which are designed and evaluated for specific domains and market mechanisms (Parsons and Klein, 2009). Moreover, it is often assumed that all agents are perfectly informed about others' actions, which is not the case in realistic settings.

This chapter presents a general framework for designing bidding strategies. As part

of Research Question 1 (*Design of Bidding Strategies*) in Section 1.2, the framework is applied in the design of a novel adaptive bidding strategy for the market-based scheduling domain called *Q-Strategy*. Bidding strategies like the *Q-Strategy* are realized to be instantiated into bidding agents in order to automate the bidding processes for consumers and providers. Section 3.1 defines the term *bidding strategy* from the classic and computational mechanism design perspectives. Section 3.2 derives design desiderata for developing bidding strategies in the market-based scheduling domain. Subsequently, Section 3.3 starts with the presentation of general frameworks for bidding in games with perfect and imperfect information. The section proceeds with the presentation of existing non-adaptive and adaptive bidding strategies, and performs an analytical comparison of selected bidding strategies according to the derived desiderata. Section 3.4 presents the *Q-Strategy* and Section 3.5 concludes with a summary of the chapter.

3.1 What is a Bidding Strategy?

Von Neumann and Morgenstern (1944) provide one of the fundamental works that initiated the study of games and economic behavior, which influenced and bundled many research fields ranging from economics and artificial intelligence to biology. In *classic economics*, it is often assumed that agents are *rational utility maximizers* acting under the assumption of *perfect information* (i.e., all agents know the actions of others at each stage of the game). The literature investigates several types of games. *Games with sequential actions* (also called extensive or tree form games¹), where each agent's possible actions and outcomes of the game are commonly represented in a tree form and each agent chooses a sequential action (also called a *pure strategy*) of the tree, based on the given stage of the game (Shoham and Leyton-Brown, 2009). In the case of *perfect information*, an agent knows all the actions of the others at any stage of the game. A utility maximizing strategy is to choose the action, which is the *best response* to actions of all other agents. The *best response* action can be unique (*pure strategy*) or include more than one possible action (*mixed strategy*). In the latter case, the agent among them is indifferent and makes a random selection with a previously chosen *probability distribution*. If all agents play a *best response* strategy, the resulting game is a *Nash equilibrium*.

¹Extensive or tree form games are visualized through a tree with nodes, edges and leaves. A node represents a choice of one of the agents, an edge represents a possible action and the leaves represent the outcomes for each player.

Other types of games are the *repeated* and *stochastic games*. In the first case, the same game is repeated over time, thus, agents learn from previous experience and adapt the actions they have selected. In contrast to *repeated games*, *stochastic games* allow the collection of normal form games to be executed repeatedly (Shoham and Leyton-Brown, 2009). Such types of games allow agents to try varying their action selection at each stage of the game in order to learn different action combinations of the game.

In real scenarios (e.g., eBay, stock exchanges), the number of agents is large and the agents have private information (e.g., valuation and reward functions), which is not known to the others at each stage of the game (also known as games with *imperfect information*). *Bayesian games* are such games with *imperfect information*, where agents face uncertainty about the others' valuation and reward functions and have limited memory to store and reason the actions of past agents. A similar type of game is called *congestion games*, which describes situations where agents compete for scarce resources (e.g., bandwidth, computing instances). Such games can become infinitely large and thus reasoning efforts become complex (NP-hard).

Game theory provides well applied and common mathematical frameworks to express various game settings, possible agent strategies, their relations and interactions. Mathematical (game-theoretic) analysis of bidding strategies is often reasonable for a small number of agents, simple utility functions, and that take place under certain, often idealistic assumptions like perfect information and rational agents (Engelbrecht-Wiggans, 1980). It can be assumed that the reason lies in the technical convenience and simplicity of such (idealistic) scenarios in order to derive straightforward evidence from the outcome of the game (Shoham et al., 2007, p. 13). With an increasing number of agents and strategy space, game-theoretic analysis and reasoning with respect to the solution space is becoming complex.

Modern economics has become an interdisciplinary discipline, where more realistic and complex scenarios can be efficiently implemented and evaluated in multi-agent experiments by utilizing computing clusters, grids and clouds. As introduced in Section 2.3, *Computational Mechanism Design* focuses on more realistic and computationally tractable mechanisms, which implies that some mechanism design desiderata are sacrificed (cf. impossibility result of Myerson and Satterthwaite (1983)). This work focuses on bidding agents and strategies for online market-based scheduling of computing services. The resulting online allocations of consumer to provider requests is not as optimal as an off-line optimization algorithm, cf. "Price of Anarchy" (Koutsoupias and Papadimitriou, 2009). Furthermore, this work investigates complex scenarios where consumer and provider agents compete in homogeneous and heterogeneous

scenarios of varying bidding strategies and different proportions thereof. In contrast to the classic game-theoretic definition of a bidding strategy, this work examines the term bidding strategy from an algorithmic perspective. In *Computational Mechanism Design*, bidding strategies are elaborated as algorithms of multi-functional decision steps, containing steps like information gathering (shared data of other agents' bids and clearing prices), data aggregation, prediction and optimization techniques, as well as bid generation. Furthermore, adaptive bidding strategies include additional steps like outcome analysis and learning from past experience at each stage of the game.

Instantiated into software agents, such bidding strategies automate the bidding processes for consumers and providers. In laboratory experiments, Das et al. (2001) showed that bidding agents (software agents applying algorithmic bidding strategies) outperform humans. The bidding agents have computational advantages over humans since data can be found, parsed and aggregated in near real-time, e.g., transactions are executed in milliseconds. Furthermore, bidding agents implement sophisticated optimization algorithms with well-defined rules and decisions. Unlike humans, bidding agents “don't get distracted”, are not indifferent between decisions and do not suffer from “auction fever” (Greenwald et al., 2003; Ku et al., 2005).

3.2 Design Desiderata

The design process of bidding strategies is a challenging task, which highly depends on the environment in terms of the target domain (here computing services), the type of market mechanism, constraints that should be considered and the actions of the other participating agents (Lin and Kraus, 2010). The characteristics of the environment define the scope for designing the bidding strategies. When designing bidding strategies for realistic settings, a clear methodology is required. The following desiderata focus on strategy design for the market-based scheduling domain.

Desideratum 1 <Automating Bidding Processes>

Bidding strategies must allow the automation of decision making processes when bidding for computing services (Windsor et al., 2009; Iyer and Huhns, 2009).

To achieve this, bidding strategies have to be designed according to the rules of the target market protocol like market type (single-sided vs. double-sided auctions), the timing of bids, offline vs. online matchmaking and information revelation (Wurman et al., 2001). Consumers and providers follow well-defined goals for their applications

and transaction objects (Iyer and Huhns, 2009; Paurobally et al., 2007). Therefore, bidding strategies have to model the transaction objects explicitly, and the goals that are to be maximized need to be configured.

Desideratum 2 <State Representation>

Bidding strategies must model the environment in which they act and interact explicitly.

From the perspective of a consumer or provider, an environment consists of *endogenous* and *exogenous* variables. Therefore, a bidding strategy designer has to decide which variables the state model is to take, *endogenous*, *exogenous*, or a mixture of both. The endogenous state of an agent expresses local states like an agent's tasks, goals, believes, desires or intentions. Examples of an endogenous state representation are those that express statements like "application X is ready for execution and needs a computing service Y" or "computing service Y is idle and ready to be offered." Such intentions can be mapped to local actions and executed by an agent. Examples of exogenous state variables are those for modeling shared information about other agents' bids and clearing prices, as well as information about the rules of the target market mechanisms (Wurman et al., 2001). Furthermore, monitoring data of an application execution on external computing services is also defined as exogenous state information. The data of the exogenous state representation is used to improve the decision making processes of the bidding strategy's endogenous model.

Desideratum 3 <Action Representation>

Bidding strategies must model the transition of states into actions, which are executed from the agents in the environment and affect their outcome.

Designers of bidding strategies must specify a policy for how actions are to be generated, according to the given state of the environment. Such policies are expressed through functions using the variables of the endogenous and exogenous states. Actions can be simple, like the generation of random numbers; they can also be complex and composed of multiple sub-tasks like information gathering, technical demand estimation, valuation estimation and bid generation.

Desideratum 4 <Goal Representation>

Bidding strategies must model the goals for trading transaction objects explicitly.

In order to automate bidding processes, the agents have to know their goals, and these have to be explicitly defined in their bidding strategies. In economics, goods

and services are compared ex-ante and selected according to a *utility function* (also called *expected utility function* if some of the parameters are unknown and have to be estimated). The same *utility function* is used for the ex-post evaluation of choice satisfaction. *Utility functions* usually have positive values and express the degree of satisfaction. In the market-based scheduling domain, consumer and provider goals can become complex and expressed with multiple attributes, so values may have a negative prefix, even if they express high satisfaction. Therefore, this work prefers to use the term *scoring function*.

Desideratum 5 ‹Adaptive Bidding in Imperfect Markets›

Bidding strategies must support the bid generation processes in markets with incomplete information and strategic heterogeneous agents.

In real settings, consumers and providers do not share their true valuations and bidding strategies with each other. This is also the case in strategy-proof mechanisms where truthful bidding is a dominant strategy (Rothkopf, 2007). Rothkopf (2007) discusses budget constraints and varying business models as practical reasons that prevent consumers and providers from revealing their true valuations (Rothkopf, 2007). This means that consumer and provider agents will behave strategically in order to achieve their goals under their budget and business model constraints. Moreover, at the beginning, it is difficult for consumers to have a clear idea of their technical requirements and economic preferences with regard to their applications. However, they can improve and refine their setups iteratively after carrying out a detailed exploration of and measuring performance on the computing infrastructures of different providers.

Furthermore, bidding strategies have to be able to deal with asymmetric market information, considering the fluctuating supply and demand of prices, and the quality and number of traded services (Vytelingum et al., 2008). The application of machine learning techniques can help aggregate past experiences to estimate the decision values under conditions of uncertainty (Anthony and Jennings, 2003; Gomes and Kowalczyk, 2007; Tesauro, 2007).

Desideratum 6 ‹Bidding in a Market-Based Scheduling Domain›

Bidding strategies for market-based scheduling must model the endogenous and exogenous characteristics of the specific domain and allow the decision and description variables to be mapped to the communication protocols of the system.

In contrast to financial markets, bidding languages for market-based scheduling include technical attributes for the description of required or provided computing ser-

vices. Part of these technical attributes may be identified as key performance indicators and included in the scoring function. Therefore, bidding strategies not only automate bid generation processes, but also provide the opportunity to adjust and “learn” the values of the identified key performance indicators. For example, a scoring function may include the minimization of completion time as part of its definition. The bidding strategy may increase the memory usage or number of CPUs if there is a correlation between completion time and memory usage or the number of CPUs (Smith et al., 1998; Ali et al., 2004).

3.3 Existing Bidding Strategies

The design of bidding strategies depends on the rules of the target market mechanism and whether all agents are perfectly informed or not. In the case of perfect information, all agents have the same data view of the system as other agents and at each stage of the game, i.e., the market provides complete information to all agents about the intentions of others – bids for the demanded and supplied computing services and clearing prices. In the case of imperfectly informed agents, at least one agent of the game is less informed than the others, i.e., the agent is not aware of something relevant to the transaction, which other agents know. An example of such asymmetries are the trading relations between suppliers and consumers when there is uncertainty about the value of the traded computing service (McAfee and McMillan, 1987). This section presents a summary of bidding strategies that are implied in markets with perfect and imperfect information.

3.3.1 Bidding in Games with Perfect Information

A well-known result in game theory is that every (finite) perfect information game in extensive form has a pure strategy Nash equilibrium. This result is due to the fact that each agent knows the actions of all others and the agents choose actions that are the best responses to all other agents’ actions at each stage of the game (Shoham and Leyton-Brown, 2009).

Formally, suppose that $s_{-i} = (s_1, \dots, s_{i-1}, s_{i+1}, \dots, s_n)$ is the strategy profile of all agents other than i and s_{-i} does not contain the bidding strategy s_i . If other agents commit to play s_{-i} at a specific stage of the game, a utility maximizing action of agent i is to determine the best response to s_{-i} .

Definition 3.3.1 (Best Response Strategy). *The best response strategy, s_i , of agent*

i with a scoring function u_i , to the strategy profile s_{-i} is the strategy $s_i^* \in S_i$ such that $u_i(s_i^*, s_{-i}) \geq u_i(s_i, s_{-i})$.

Definition 3.3.2 (Nash Equilibrium). *Strategy profile $s = (s_1, \dots, s_n)$ is a Nash equilibrium if, for all agents i , s_i is the best response to s_{-i} .*

The concept of Nash equilibrium provides a stable strategy profile, in which no agent wants to deviate from the best response strategy s_i^* , if other agents play s_{-i} .

To calculate Nash equilibria, the literature proposes algorithms that are based on *backward induction*. *Backward induction* calculates Nash equilibria for each sub-game (tree) of the overall game tree using a depth-first traversal search. *Such algorithms run well for small numbers of agents and strategies, but become unfeasible in practical settings because of the exponentially increasing space of the tree game in terms of agents and their adopted strategies.*

In contrast to extensive form games, perfect information repeated or stochastic games may have more than one Nash equilibrium due to the learning processes of the various playing agents and the resulting new action. For example, the classic *Rock-Paper-Scissors* game is a zero-sum game, where the best response strategy of each player is to uniformly select an action from three of the actions possible. In *Rock-Paper-Scissors* tournaments, the winners are never Nash equilibrium players (Shoham et al., 2007). The winner of one of the past tournaments said, “I read the minds of my competitors and figure out what they are thinking. I don’t believe in planning your throws before you meet your opponent” (Shoham et al., 2007). The strategy space in a repeated game (or more generally a stochastic game) is immense because it includes all the mappings from past history to mixed strategies in the stage game. In such complex games it is not reasonable to expect that agents can explore the entire strategy space, but learn to achieve their own goals. Therefore, in such games, formal Nash equilibrium analysis plays a minor role, if any at all.

3.3.2 Bidding in Games with Imperfect Information

Games with perfect information assume that all agents know the actions of others at each stage of the game, as well as the actions of others that led to the current stage. In a more realistic and competitive market setting, there is asymmetric information about other agents’ intentions and their rewards. An agent reward depends on its own preferences, the actions of other agents and the quality of the traded transaction objects. Such a setting, where indivisible objects are traded and there is uncertainty

about other agents' private values, is called the *Independent Private Values Model* (IPV) (Milgrom and Weber, 1982). In this model, each agent is assumed to be risk neutral and each agent has his own private value estimate for the traded transaction object. Furthermore, it is assumed that agents compete among each other in a non-cooperative game, with each of them maximizing his own utility.

The following theoretic model defines the class of *imperfect information games*, which is based on the definition in (Shoham and Leyton-Brown, 2009, p. 165).

Definition 3.3.3 (Imperfect Information Game). *An Imperfect Information Game is represented through the tuple $G = (N, \Omega, A, R, \rho, S, M)$, where:*

- N is the set of all consumer and provider agents;
- $\Omega = (\Omega_1, \Omega_2, \dots)$ is the set all agents' states, where $\Omega_i = (\omega_{i,1}, \omega_{i,2}, \dots)$ represents the set of states of agent $i \in N$;
- $A = (A_1, A_2, \dots)$ is the set of all agents' actions, where $A_i = (a_{i,1}, a_{i,2}, \dots)$ represents the set of actions of agent $i \in N$;
- $R : \Omega_i \times A_i \rightarrow \mathbb{R}$ is the scoring function for calculating the score (reward) for each executed action $a_{i,k} \in A_i$ and each $\omega_{i,k} \in \Omega_i$ of agent $i \in N$;
- $\rho : \Omega_i \times A_i \rightarrow A_i$ specifies the transition probability function of taking an action $a_{i,k} \in A_i$ in state $\omega_{i,k} \in \Omega_i$;
- $S = (s_1, s_2, \dots)$ is the set of all agents' bidding strategies, where s_i is the selected bidding strategy of agent i e.g., Q-Strategy, ZIP, GD etc;
- $M = (M_1, M_2, \dots)$ is the set of all agents' messages, where $M_i = (m_{i,1}, m_{i,2}, \dots)$ represents the set of messages of agent $i \in N$.

Definition 3.3.3 extends the definition in (Shoham and Leyton-Brown, 2009, p. 165) with the addition of the S concept, which explicitly models the set of different (algorithmic) bidding strategies to the set of available agents N of the game setting G .

An agent i in an *imperfect information game* can select a bidding strategy s_i , based on her or his experience in past and similar games. Let $H_i = (h_{i,1}, h_{i,2}, \dots)$ be the agent i 's history with a memory length of T units of time.

Definition 3.3.4 (Classic Markov Strategy). *A Markov strategy $s_i : \Omega_i \times H_i \rightarrow A_i$ of an agent i returns the action $a_k \in A_i$ with the highest probability of being successful for the given state $\omega_k \in \Omega_i$ and history H_i .*

The utility (score) of an agent, given other agents' strategies, is a common indicator for games with incomplete information.

Definition 3.3.5 (Expected Utility). *An agent i 's expected utility with a bidding strategy s_i and other agents' strategies s_{-i} is defined as:*

$$E_i(s_i, s_{-i}) = \sum_{a_k \in A_i} \left(u_i(a_k) \prod_{a_{-k} \in A_{-i}} s_{-i}(a_{-k}) \right) \quad (3.1)$$

In an *imperfect information game*, the bidding strategies of other agents are now known to agent i , thus, agent i calculates its expected utility based on the public information of other agents' actions A_{-i} . Thus, in contrast to the classic definition, other agents' strategies s_{-i} are not known and agent i calculates the expected utility from its own perspective:

$$E_i(s_i, s_{-i}) = \sum_{a_k \in A_i} \left(u_i(a_k) \prod_{a_{-k} \in A_{-i}} u_i(a_{-k}) \right) \quad (3.2)$$

Definition 3.3.6 (Best Response Strategy). *The set of agent i 's best responses to other agents' strategies s_{-i} is defined as:*

$$BR_i(s_{-i}) = \arg \max_{s'_i \in S_i} E_i(s'_i, s_{-i}) \quad (3.3)$$

Definition 3.3.7 (Bayes-Nash Equilibrium). *A Bayes-Nash equilibrium is a strategy profile s that for all i , $s_i \in BR_i(s_{-i})$*

The difference of the Bayes-Nash equilibrium to the Nash equilibrium in normal form games is that in *imperfect information games* the agents do not know the independent private values of other agents, thus they try maximize their expected utility based on the public information available on other agents. In order to calculate the best response in a *imperfect information game*, agent i must know the current actions A_{-i} of the others agents for the state Ω_i .

In general, bidding strategies for *imperfect information games* can be classified as *non-adaptive* when the generated bid does not depend on past and current market information (e.g., bids, offers, clearing prices), and as *adaptive* when the generated bid depends on the available market information. In the following sections the non-adaptive and adaptive strategies that are usually applied are introduced.

3.3.3 Non-Adaptive Bidding Strategies

Non-adaptive bidding strategies do not require any public information from the market mechanism to generate the bids. Such bidding strategies are applied in markets where either there is a well-known dominant strategy, and therefore no point of adaption, or in markets with a “high degree of uncertainty”, where agents behave randomly in some ways, c.f “random walk” in financial markets (Campbell et al., 1997).

In this context, *Truth-Telling* is the simplest strategy, where the reported bid to the market equals the independent private value of the agent. This strategy constitutes the best response in a strategy-proof mechanism, where truthful bidding is a dominant strategy, i.e., it is a utility maximizing strategy regardless of what other agents bid. Market mechanisms, which are strategy proof reduce the strategy space of agents that do not need to perform game-theoretic analysis or counter-speculation of others. The existing literature often focuses on the investigation of strategy-proof mechanisms in various application scenarios (Kalagnanam and Parkes, 2004). However, market mechanisms, such as bid generation, communication costs, and winner determination, which are theoretically shown to be strategy proof, actually present NP-hard problems in practice. Moreover, it is often assumed that the independent private values of consumers and providers are not constrained by any budgets or endogenous constraints (Rothkopf, 2007).

Another non-adaptive bidding strategy is the *Zero-Intelligence* (ZI), which is also called the *Zero-Intelligence Constraint* (ZI-C). ZI or ZI-C is one of the first and simplest algorithmic bidding strategies evaluated in laboratory experiments against human agents (Gode and Sunder, 1993). The ZI-C strategy agents submit (uniformly distributed) random bids based on the budget constraints of consumers and providers – consumers do not buy above their independent private values and providers do not sell below their operating costs.

The authors promote this strategy based on the assumption that the efficiency of a double auction depends on market design rules rather than the learning effects of agents. Furthermore, it postulates that it is not possible to predict the trading behavior of agents because of their heterogeneity in terms of expectations, preferences, risks and endowments. The experiments showed that budget constraint is sufficient to achieve efficient outcomes that are comparable to those of the human agents. The ZI-C strategy is a simple strategy, which can be applied in computing service markets where there is high uncertainty about the prices and technical requirements.

3.3.4 Adaptive Bidding Strategies

Adaptive bidding strategies are designed to work with the public data available, which is provided by the market mechanisms. The following sections will present two commonly applied benchmark strategies for the *Continuous Double Auction* (CDA), which is the target market model of this work. Furthermore, other adaptive bidding strategies that are valuable for this research are discussed, but are designed for a different \widehat{CDA} type than the online CDA type presented in Section 2.3.5.

Zero Intelligence Plus Strategy

Cliff and Bruten (1997) developed an adaptive bidding strategy called Zero Intelligence Plus (ZIP). ZIP agents have been widely explored and constitute a popular benchmark for agents trading on Continuous Double Auctions (Das et al., 2001). Central to the ZIP agent's performance is the rule for updating the profit margins of providers (Algorithm 3.3.1) and consumers (Algorithm 3.3.2), which is the difference between the agent's independent private value and the bid. In the absence of any information, the strategy is initialized with a low bid for the trading object in comparison to its true valuation. The profit margin is automatically updated according to a rule, which is based on the market information received for the clearing prices and bids of other agents. The most recent bid of another agent is denoted by q_{-i} . Increasing the profit margin μ_i , raises the agent i 's bid q_i in the case of a seller and lowers q_i in the case of a consumer.

Algorithm 3.3.1: ZIP: PROVIDER'S PROFIT MARGIN UPDATE RULE(q_{-i}, μ_i, q_i)

comment: Compute provider's profit margin μ_i , based on
the last market information q_{-i} and the provider's bid q_i

if q_{-i} was a match

then	{	if $q_i \leq q_{-i}$ then { <i>raiseProfitMargin</i> (μ_i) if q_{-i} was a consumer bid and $q_i \geq q_{-i}$ then { <i>lowerProfitMargin</i> (μ_i)
-------------	---	--

else if q_{-i} was not a match

then	{	if q_{-i} was a provider bid and $q_i \geq q_{-i}$ then { <i>lowerProfitMargin</i> (μ_i)
-------------	---	--

Algorithm 3.3.2: ZIP: CONSUMER'S PROFIT MARGIN UPDATE RULE(q_{-i}, μ_i, q_i)

comment: Compute consumer's profit margin μ_i , based on
the last market information q_{-i} and the consumer's bid q_i
if q_{-i} was a match
then $\left\{ \begin{array}{l} \text{if } q_i \geq q_{-i} \\ \quad \text{then } \{ \text{raiseProfitMargin}(\mu_i) \} \\ \text{if } q_{-i} \text{ was a provider bid and } q_i \leq q_{-i} \\ \quad \text{then } \{ \text{lowerProfitMargin}(\mu_i) \} \end{array} \right.$
else if q_{-i} was not a match
then $\left\{ \begin{array}{l} \text{if } q_{-i} \text{ was a consumer bid and } q_i \leq q_{-i} \\ \quad \text{then } \{ \text{lowerProfitMargin}(\mu_i) \} \end{array} \right.$

ZIP's relationship to the generated bid, valuation and profit margin of an agent i for a trading object ω_i is represented with the following rule:

$$\begin{aligned} q_i &= v_i(1 + \mu_i) \\ \text{with } \mu_i &\in [0, \infty] \text{ in the case of provider} \\ \text{with } \mu_i &\in [-1, 0] \text{ in the case of consumer} \end{aligned} \quad (3.4)$$

The rules for raising and lowering the profit margins of consumers and providers are altered dynamically based on whether the last signal was a consumer or provider bid and whether the bid was a successful match or if it is still unmatched. The profit margin for the upcoming bid is calculated according to the following update rule:

$$\mu_i = \frac{q_i + \Delta_i}{v_i} - 1 \quad (3.5)$$

Δ_i is the Widrow-Hoff delta value, which is calculated with the individual agent's i learning rate β_i and the target price τ_i :

$$\Delta_i = \beta_i(\tau_i - q_i) \quad (3.6)$$

A ZIP bid is generated with the following stochastic function:

$$\tau_i(t) = R_i q_{-i} + A_i \quad (3.7)$$

Here, R_i is a randomly generated coefficient, which sets the target price relative to the current bid q_{-i} with ranges of $R \in [1.0, 1.05]$ for price increases and $R \in [0.95, 1.0]$ for price decreases (Cliff and Bruten, 1997). A_i is a random value alteration variable,

which is set to be uniformly distributed over $A_i \in [0.0, 0.05]$ for price increases and $A_i \in [-0.05, 0.0]$ for price decreases. In Das et al. (2001), experiments show that ZIP agents perform better than (non-expert) human traders on CDA markets. Numerical experiments show that adopting a ZIP strategy in markets dominated by other kinds of agents results in an increased profit when the dominating agents are ZI, Kaplan or *GD* agents (Gjerstad and Dickhaut, 1998). The ZIP strategy performs well with fluctuating demand and supply and converges quickly in CDA markets.

In summary, ZIP is an adaptive, dynamic and *price competitive* strategy, i.e., it adapts quickly to changing market conditions in terms of bids and clearing prices with the aim of maximize the agent's profit. Such a strategy can be applied in computing service markets where there is public information available about other agents' bids and market clearing prices. A drawback of the ZIP strategy is that it is not applicable in markets with missing or sparse public information about the price signals of other agents. Furthermore, it indirectly implements a fixed short-term objective of profit maximization. Nevertheless, this bidding strategy is one of the benchmark strategies for CDAs in the current literature (Schvartzman and Wellman, 2009; Vytelingum et al., 2008; Tesauro and Das, 2001).

Gjerstad-Dickhaut Strategy

Another benchmark bidding strategy for CDA markets is the *Gjerstad-Dickhaut* strategy, which is also called *Heuristic Belief Learning* (Gjerstad and Dickhaut, 1998; Gjerstad, 2003). Similar to ZIP, *GD* also requires public information about successful and unsuccessful consumer and provider bids in order to calculate competitive bids. *GD* generates bids based on historic information and belief functions for consumers and providers. The *GD* strategy is also designed to be price competitive. It has a profit maximizing scoring function, as well as the ability to respond quickly to changing market conditions of supply and demand.

Compared to ZIP, this mechanism is memory based, i.e., it maintains a history H_i of the last T bids and clearing prices. The so-called "belief" function $f(\pi)$ calculates the probability for a bid or offer to be accepted at price π .

Let $APB(\pi)$ be the set of *provider bids* that have been *accepted* at a price greater than or equal to π and $UPB(\pi)$ be the set of *unaccepted provider bids* at a price greater than or equal to π . Let $CBL(\pi)$ be the total set of *consumer bids* lower than or equal to π . Then, the provider j belief function is defined as:

Definition 3.3.8 (Provider's Belief). $f_j(\pi) = \frac{|APB(\pi)| + |CBL(\pi)|}{|APB(\pi)| + |CBL(\pi)| + |UPB(\pi)|}$

The intuition of Definition 3.3.8 is that if a provider bid has been rejected at $o' < o$, it will also be rejected at o and vice versa for the consumer case, Definition 3.3.9.

Analogous to the consumer case, let $ACB(\pi)$ be the set of *consumer bids* b that have been *accepted* at a price lower than or equal to π and $UCB(\pi)$ be the set of *unaccepted consumer bids* at a price lower than or equal to π . Let $CBL(\pi)$ be the total set of *provider bids* greater than or equal to π . Then, the consumer i belief function is defined as in following:

Definition 3.3.9 (Consumer's Belief). $f_i(\pi) = \frac{|ACB(\pi)| + |CBL(\pi)|}{|ACB(\pi)| + |CBL(\pi)| + |UCB(\pi)|}$

The timing of bids is another strategic variable in GD. Bids are randomly delayed over time according to the exponential distribution and the distribution's *rate parameter* depends on the maximum expected reward of the bid and the length T of the trading period. As such, the choice of exponential distribution is motivated by the fact that *GD* is designed to be exploited in mechanisms where there is no common knowledge about other agents' rewards and independent private values.

Spline interpolation ensures that the belief functions are monotonically decreasing for the provider case and monotonically increasing for the consumer case. The provider j and consumer i bids are calculated according to the following optimization problems of the expected reward:

$$\max \left(E(v_j, \pi) = (\pi - v_j) f_j(\pi) \right) \quad (3.8)$$

$$\max \left(E(v_i, \pi) = (v_i - \pi) f_i(\pi) \right) \quad (3.9)$$

In summary, the *GD* strategy implements an adaptive model for bid generation, based on observed public market information, as well as agents' independent private value. The choice of an action (bid) depends on the agent's belief function of past and current successful and unsuccessful bids. As with ZIP, the *GD* strategy is also designed to be price competitive.

A drawback of the *GD* strategy is that its bid generation process also requires public information about other agents' bids and clearing prices. Like the ZIP strategy, GD is also designed to maximize profit.

General Learning Models

Learning algorithms have been applied in many fields for solving complex problems in single or multi-agent environments such as autonomous bidding agents (TAC game

family), machine perception and cognition, autonomic computing², autonomous vehicles, natural language processing, search engines and fraud detection (Kephart and Chess, 2003; Tesauro, 2007; Vulkan, 2003; Stone, 2007a).

In general, learning algorithms are classified into supervised, unsupervised and reinforcement learning algorithms. *Supervised learning (SL) algorithms* are applied in cases when the sources of input and output data are available to the agents. The agents can perform statistical (regression) analysis of the existing data in order to “train” their parameter sets or bidding strategies. The ATTac-2001³ agent is a successful agent of the *Trading Agent Competition*, which applied SL algorithms in order to estimate the probability distributions of prices when competing against other agents in the TAC game (Stone et al., 2003). Vorobeychik et al. (2007) applied SL algorithms to learn reward functions and estimate successful strategies through regression analysis of training sets. However, these strategies are applicable in scenarios when the agents have access to the actions and responses of all other agents. Furthermore, the ATTac-2001 agent was trained ex-ante with the data of the previous TAC games. In a real scenario for market-based scheduling, one needs bidding strategies, which are able to maximize owners’ scoring functions online and are based on the available (partial or aggregated) market information of other agents. Moreover, in a real scenario, it is not expected that other agents will report their true valuations and adopted algorithmic bidding strategies.

In contrast, *unsupervised learning (UL)* algorithms perform statistical analysis on given (historic) input data, without given reference to any output data. UL algorithms aim at discovering new structures (clusters, similarities), patterns or relationships between variables in the input data, based on the objective function maximization of similarity patterns. Target applications of UL algorithms are in the field of data mining, in which closely related objects are clustered according to the similarity of the given criteria (Tesauro, 2007). Kiselev and Alhajj (2009) proposed an online unsupervised learning approach for the hierarchical clustering of messages exchanged between agents in a decentralized and dynamic multi-agent environment, with the aim of resource allocation problems through negotiations. Another application of UL

²“Self-*” tasks like dynamic allocation of bandwidth, memory, threads, and logical partitions; online performance tuning of system control parameters for Web servers, operating system and database parameters.

³ATTac-2001 assumes that public information exists with regard to other (software) bidding agents and their strategies (actions) from past rounds in TAC games with many rounds. The ATTac-2001 agent was “tuned” in each round based on the available input and output data from past rounds and knowledge about the participating agents in the coming rounds (Stone et al., 2003).

algorithms is to cluster technical requirements into similar *classes* and to calculate expected rewards for “similar” future actions, that are based on past experience data. Moreover, consumers do not have the expertise to determine technical requirements, however, they may give estimated requirements for their applications on a more abstract level. UL algorithms can be applied to reason the historic (public or private) information available of “similar” applications and their rewards and to infer an initial detailed technical specification from the “abstract” consumer statements. Thus, clustering algorithms can map such abstract benchmark concepts to more detailed description of technical requirements.

Unlike supervised and unsupervised learning, where agents are taught with examples of input and output data in order to derive new actions, *reinforcement learning* (RL) collects the data through interaction with the environment and thus it is more suitable for online sequential decision making problems (Sutton and Barto, 1998; Tesauro, 2007). The goal of RL is to learn state-action pairs from delayed rewards, which are received from the environment from the execution of actions. The potential of RL is largely unstudied for systems management applications. A commonly studied application of RL is learning about effective actions “in the absence of explicit system models, with little or no domain-specific initial knowledge” (Tesauro, 2007). The advantages of reinforcement learning over supervised learning are that i) “there is no requirement for a skilled human to provide training examples;” ii) “the exploration process allows the agent to become competent in areas of the state space that are seldom visited by human experts and for which no training examples may be available” (Dearden et al., 1998). Moreover, Dearden et al. (1998) stated that “to ensure a more robust behavior across the state space, exploration is crucial in allowing the agent to discover the reward structure of the environment and to determine the optimal policy. Without sufficient incentive to explore, the agent may quickly settle on a policy of low utility simply because it looks better than leaping into the unknown.” “A good exploration method should balance the expected gains from exploration against the cost of trying possibly suboptimal actions when better ones are available to be exploited.” Good approximative solutions for problems in reinforcement learning are generally hard to find and may only be found in specific cases like in “the so-called bandit problems in which the environment has a single state, several actions, and unknown rewards” (Dearden et al., 1998).

The major difference of reinforcement learning (RL) in comparison to supervised and unsupervised learning is that RL has to explore the environment in order to learn the “best” actions with a given state. The designer of an RL algorithm has to think of good policies that achieve a good trade-off in exploring the environment and in

exploiting the learned data in new state-action pairs. RL algorithms aim to mimic human learning behavior, which is described by two of the main principles found in the literature on psychological learning, the *Law of Effect* and the *Power Law of Practice* (Sutton and Barto, 1998). The *Law of Effect* states that an action is more likely to be selected from a set of actions that have achieved the highest cumulative rewards in the past; in other words, successful actions are likely to be strengthened over time and unsuccessful actions weakened. *Power Law of Practice* assumes that the learning curve is steep at the beginning and flatten out over time. This principle states that successful and similar actions will be employed more often than others, and also postulates that recent experiences are weighted higher than past experiences (Sutton and Barto, 1998).

In general, reinforcement learning algorithms can be broken down into *Model-based learning* and *Model-free learning* mechanisms. *Model-based learning* focuses on policies to learn an opponent's strategies in order to estimate an approximated "best response" to others' actions. It is applied in environments where there is information available on other agents and their actions. However, learning opponents' strategies within a given time frame does not guarantee that the opponents will act the same way in the future.

In the case of *Model-free learning*, the agent learns "optimal" actions that perform well in a given environment and against diverse opponents based on its local states (its own requirements, preferences and scoring functions). The model-free learners do not try to estimate the opponent's strategy explicitly, but to adapt their own actions based on the observed outcome, i.e., the rewards received from the own actions (Kaelbling et al., 1996). Therefore, model-free learning mechanisms indirectly incorporate the dynamics of other agents' actions since the outcome is based on their fluctuating supply and demand. *Model-free learning* is motivated by the fact that the environment is not fully transparent to all agents, i.e., agents do not reveal their private information in terms of independent private values and scoring functions. Furthermore, a consumer agent will not always know its own preferences and may report a fuzzy specification of its technical requirements. The technical requirements can be manually adapted by the consumer or automatically adapted by the bidding agent with the aim of maximizing a given scoring function. This work focuses on *Model-free learning* approaches since the environment, participants and their actions (bids) are changing dynamically over time and especially in the computing service domain.

In order to foster the learning process and overcome complexity, Stone (2007a) summarized the techniques of *Learned Abstractions* and *Layered Learning*. *Learned Ab-*

stractions is a useful technique when classifying different objects according to common attribute-value pairs and parameter bounds. This technique enables quicker learning with a higher number of actions that belong to the abstract class of objects, rather than single objects. This learning technique can be applied in scenarios with well-known domain representations, where the human agent can manually determine the conditions for classifying the objects.

Learning-Based Bidding Strategies

Roth and Erev (1995) and Erev and Roth (1998) specified a reinforcement learning algorithm with one, three and four parameters, which are adapted in games with multiple agents. The authors' goal was to elaborate general reinforcement learning approaches, which aim to mimic human strategic behavior in games with multiple players. The three-parameter learning algorithm was later called the *Roth-Erev* or *RE strategy* and was evaluated in a Clearing House market mechanism (also called Call Market) against the *Truth-Telling*⁴ and *GD* strategies (Phelps et al., 2006). The *Roth-Erev* learning algorithm “solves a myopic stimulus-response problem of the following form: Given this profit outcome, what price should I next choose?” (Nicolaisen et al., 2001). However, the *RE strategy* does not explicitly model the preferences of the agents (Erev and Roth, 1998, p. 875).

Many of the bidding strategies in the literature have financial markets as a target scenario, such as the *Adaptive-Aggressiveness (AA)* of Vytelingum et al. (2008) and the reinforcement learning-based strategy of Schwartzman and Wellman (2009). Their bidding strategy design and evaluation methodology targets a specific variant of \widehat{CDA} , which is applied in financial markets and where the agents compete in several rounds during several days and the start and end times of a round and day is known to all agents (Friedman, 1993). Bidding strategies like *GDX* (enhanced *GD*) and the strategy of Schwartzman and Wellman (2009) use the specific time information in their decision-making process, i.e., “the expected number of bidding opportunities before the auction closes” (Tesauro and Bredin, 2002; Schwartzman and Wellman, 2009). Moreover, with some strategies like the *AA* model in \widehat{CDA} constraints such as the *spread-improvement rule*, consumer bids can be placed if they are below the current minimum consumer bid and vice versa for the provider case; with the *no-order queuing rule* as well, unsuccessful bids and offers are not queued in the order book (Vytelingum et al., 2008). Cliff (2006) also proposes an enhanced version of the ZIP strategy called “ZIP60,” which adapts 60 financial parameters with

⁴Note: Clearing-House is not a strategy-proof mechanism.

genetic algorithms in \widehat{CDA} markets. Park et al. (1996, 1999, 2000, 2004) developed the reinforcement learning-based bidding strategy called *P-Strategy*, which computes probabilities for state and reward transitions based on stochastic modeling. He et al. (2003) apply fuzzy rules and reasoning mechanisms into a heuristic bidding strategy in order to find efficient actions for a given market state based on the information available from other agents.

Reeves et al. (2005) explored bidding strategies for market-based scheduling in simultaneous ascending auctions for allocating CPU slots. They evaluated a baseline bidding strategy called *straightforward bidding*, which is also known as a “myopic best response.” Based on the current perceived bids for available slots, each agent selects a best response and utility maximizing bid to purchase a given slot. However, the authors showed that this strategy is not an efficient (dominant) strategy for ascending auctions. Moreover, the authors proposed a bidding strategy called “sunk awareness,” which assumes that other agents share their *mixed strategies* (from a game-theoretic perspective) with each other and each of the agents can perform evolutionary searches (replicator dynamics) to find an “optimal” strategy. A similar approach for evolutionary searches of bidding strategies is followed and proposed in Phelps et al. (2010b,a).

Sandholm et al. (2006) discuss models to predict the prices of computing services, and applied their methods in the *Tycoon* spot market (Lai et al., 2005). However, they stated that it is hard to predict user demand and the accuracy of the “predictions depends on the regularity of previous price snapshots and it is therefore crucial, for the results to be good, to pick a time window to study that exhibits these patterns.”

In contrast to the works discussed above, this thesis selects a type of *CDA*, which does not introduce time constraints for rounds. The *CDA* type of this work is assumed to be “infinite” and running continuously every day of the year (Chevaleyre et al., 2006, p. 11). This design choice is motivated by the fact that computing services are not storable and their usage patterns are highly dynamic (Altmann et al., 2008). Any restriction on their continuous allocation and usage will effect their economic efficiency from both, a provider and consumer perspective.

3.3.5 Analytical Comparison to the Desiderata

The state-of-the-art bidding strategies investigated have heterogeneous designs of their states, actions, goals, information requirements, target auction type and application domains. Table 3.1 shows a mapping of the elaborated bidding strategies to the common and domain-specific design desiderata derived. In addition to the design

desiderata, the table compares the bidding strategies according to the *information* required for generating their *actions* according to their *goals*, as well as their target *auction* types and *constraints* for which the strategies are designed and evaluated.

Desiderata D1 to D5 are common for the design of bidding strategies. However, they are evaluated further in the context of market-based scheduling, which is explicitly required in desideratum D6.

Automating Bidding Processes

All of the bidding strategies discussed satisfy the general desideratum D1. Specifically, the information required as input, as well as the calculation procedure of their bid generation processes are well-defined and implemented in their data aggregation and decision models. All of these strategies are instantiated into bidding agents, therefore the bidding processes of consumers and providers are automated.

State Representation

Truth-Telling and *ZI-C* are simple bidding strategies, which do not explicitly define a specific state model. However, they generate bids for a given artifact, which is traded on the market. These two models only require the valuation as information in order to generate the true or random bids. The *Straightforward Bidding*⁵ (SB) strategy requires a perfect knowledge of other agents' actions like bids, offers and prices in order to calculate the best response strategy (action) to them. This strategy is designed and evaluated for the specific *Simultaneous Ascending Auction* (SAA) (Reeves et al., 2005). The state model of SB is not explicitly defined, however, the authors model stated that the calculation of the best response is based on the prices perceived and free slots for the computing services traded. A similar, but complex approach is taken in the *Evolutionary search* strategies. With an *evolutionary search* genetic algorithms are applied to learn of other agents' actions and rewards from a stationary environment in order to calculate the best responses to them. *Evolutionary search* does not explicitly model a state; it requires full knowledge of the environment is required and having to deal with rising computation complexity with the number of agents and their actions. Therefore, *Straightforward Bidding* and *Evolutionary search* are more likely to be adopted in offline and perfect information settings (cf. *Call Market* in Phelps et al. (2010b)), than in online imperfect markets.

⁵Myopic Best Response.

Table 3.1: Mapping of the desiderata for strategy design to existing bidding strategies. D1 – Automating the Bidding Processes. D2 – State Representation. D3 – Action Representation. D4 – Goal Representation. D5 – Adaptive Bidding in Imperfect Markets. D6 – Bidding in the Market-Based Scheduling Domain. ✓ indicates explicit support ○ indicates that no explicit information was found or the desideratum is not met. ● indicates partial support.

Strategy	D1	D2	Information	D3	D4	Goal	D5	D6	Auction	Constraint
Truth Telling	✓	●	valuation	✓	●	bid truthful	○	○	<i>SPM</i>	no
Zero Intelligence	✓	●	valuation	✓	●	bid random	○	○	<i>CDA</i>	no
Straightforward Bidding	✓	●	perfect	✓	✓	max. profit	●	●	<i>SAA</i>	others' actions
Evolutionary Search	✓	●	perfect	✓	✓	max. profit	●	○	<i>Call Market</i>	others' actions
Roth-Erev	✓	●	(perfect)	✓	✓	best price	●	○	<i>Call Market</i>	(others' actions)
P Strategy	✓	✓	perfect	✓	✓	max. profit	●	○	<i>Call Market</i>	others' actions
Gjerstad-Dickhaut-X	✓	✓	perfect	✓	✓	max. profit	●	○	\widehat{CDA}	others' actions
Adaptive-Aggressiveness	✓	✓	perfect	✓	✓	max. profit	●	○	\widehat{CDA}	end of round/day others' actions
Schwartzman-Wellman	✓	✓	perfect	✓	✓	max. profit	●	○	\widehat{CDA}	end of round/day others' actions
Zero Intelligence Plus	✓	●	perfect	✓	✓	max. profit	●	○	<i>CDA</i>	others' actions
Gjerstad-Dickhaut	✓	●	perfect	✓	✓	max. profit	●	○	<i>CDA</i>	others' actions
This work	✓	✓	local states reward	✓	✓	max. score	✓	✓	<i>CDA</i>	no

As adaptive strategies, *Roth-Erev* (RE) and *P Strategy* are designed to predict prices in financial markets based on other agents' actions. Generally, the RE strategy applies the Reinforcement Learning (RL) model directly, the "one parameter" RE specifies the so-called *propensity*, which is actually the reward parameter in an RL model. The "three parameter" RE also defines *experimentation* and *recency*, which represent the *epsilon-greedy* exploration policy and *discount factor* of an RL model. Finally, the "four parameter" RE model calculates an expected value for an action based on the available (perfect) information of other agents' actions in the environment (Erev and Roth, 1998). The model of the RE state is not explicitly defined in terms of attributes and a state transition function.

P Strategy is based on stochastic modeling and the first strategy in the table, which explicitly defines a state model and a state transition function. The *P Strategy* state model contains endogenous and exogenous variables. The endogenous model contains local information about the bidder's action history and the valuations of the transaction objects. The exogenous model contains shared market information like the number of standing bids and prices, their probability distributions, the arrival rates of bids and the status of the auction (Park et al., 1996). However, in later works, only some of these variables are explicitly considered in the *P Strategy* model and evaluation scenarios (Park et al., 1999, 2000, 2004). The *P Strategy* performs a form of evolutionary search on the state information again and thus is computationally intractable for settings of multiple agents and actions (Park et al., 2004).

The *Gjerstad-Dickhaut-X* (GDX) bidding strategy extends the original *Gjerstad-Dickhaut* strategy with a dynamic programming technique (Tesauro and Bredin, 2002). The GDX model a state by an agents holdings, the holdings' transition probabilities and the remaining time until the end of a trading day. The holdings' transition probabilities are estimated based on the past actions of other agents and the "belief function" from the original *Gjerstad-Dickhaut* strategy.

Moreover, the state transition function calculates a time forecast of when the agent should bid by optimizing the long-term discounted profits. This is realized over a dynamic programming model and based on the available market information about the auction round and time of day.

The *Adaptive-Aggressiveness* (AA) strategy is a two-layer bidding strategy with short and long-term learning components. The short-term learning component combines ideas from the ZIP and GDX bidding strategies, where the long-term learning component adapts a so-called *intrinsic shape parameter*, which is used to update the so-called *aggressiveness factor*. The latter is used from the short-term learning component to decide when and how much to bid (Vytelingum et al., 2008). The aim

of the short-term learning component is to detect price fluctuations (market shocks) based on the observed market information of other agents' actions, bids and prices. Like the GDX strategy, the AA strategy is designed to consider the time constraints of a \widehat{CDA} mechanism in terms of end of round and end of day. Furthermore, the \widehat{CDA} implements a *spread-improvement rule*.

Schwartzman and Wellman (2009) propose an evolutionary search model for finding "optimal" bidding strategies through reinforcement learning. They suggest applying empirical game-theoretic analysis to accumulate data about the performance of the selected agents' actions through Monte Carlo simulations. Their model applies regression analysis techniques through pre-generated data and it is applicable in static environments with less fluctuations in demand and supply. Like with the AA strategy, the target auction type in Schwartzman and Wellman (2009) is \widehat{CDA} , where the time constraints and learning effects are incorporated in the state of the evolutionary search model. The state model is complex; it contains time-related data (total time elapsed in the current trading period, and time elapsed since the last trade), transaction object-related data (number of units left to trade) and statistical (regression analysis) data of agents' actions. Statistical data include moving averages of recent trades, profits, probability information about recent bids and prices, as well as the valuation of the transaction object. However, the model in Schwartzman and Wellman (2009) is not applicable in practical settings since in real settings the number of agents and their actions is high, and the target environment is non-stationary with dynamic supply and demand. Moreover, the effort to find "optimal" bids in stationary and fixed settings is huge and the authors "provide no recipe for this issue beyond relying on domain knowledge, creativity, and plenty of patience" (Schwartzman and Wellman, 2009).

Similar to GDX and RE, the ZIP and GD strategies do not model their states explicitly, but their in CDA and \widehat{CDA} settings. ZIP and GD are evaluated in both auction types since they do not directly incorporate time constraints, but explicitly define how the bids are generated without these constraints. Moreover, ZIP and GD are commonly applied benchmarks in the literature and for the bidding strategies discussed in the previous paragraph (Tesauro and Bredin, 2002; Vytelingum et al., 2008; Schwartzman and Wellman, 2009).

Action Representation

Desideratum D3 is satisfied by all of the analyzed bidding strategies in Table 3.1. The common action for all of these strategies is the generated bid, which is submitted to the market. The representation of their actions differ in the data and decision models

they apply to generate the bids. Most of them use perfect market information to calibrate the decision parameters. Statistical techniques, evolutionary search and dynamic programming are applied to predict when and what to bid.

The RE strategy applies the epsilon-greedy selection policy to switch between exploration and exploitation modes when generating the bids. In the exploration mode, the RE strategy chooses a random action as a bid, and in the exploitation mode it selects an action, which was successful in the past since this action is also more likely to be successful in the future. The RE strategy implements this in a propensity matrix, which stores the explored actions and their propensity values (discounted rewards). For each executed action, the RE strategy updates its propensity value from the received reward. In the exploitation phase, the RE strategy performs a lookup to select the action with the highest propensity.

The *P strategy* uses shared public information and performs statistical analysis to estimate an action, which maximizes the agent's profit. The probabilities for success and failure are reinforced in the model from other agents' actions, i.e., bids and prices. However, these computations are timely expensive with respect to the number of agents and their actions, thus, this model is not practical in online settings (Park et al., 2004). *Straightforward Bidding* and *Evolutionary search* perform optimization searches of the available market information to find the best response strategies. Like with ZIP and GD, GDX, AA and Schwartzman and Wellman (2009), the decision variables are adapted from other agents' actions (perfect information). ZIP updates the profit margin on each bid or price received, which is used when bids are generated. GD uses a "belief function," which calculates the probability for each candidate bid to be allocated and selects the bid with the highest probability. GDX calculates state-transition probabilities for when and what to bid, which is based on a dynamic programming model of the public information. AA updates an "aggressiveness" parameter, which is part of the decision function for when and what to bid. The Schwartzman and Wellman (2009) strategy performs an evolutionary search using agents' actions to find an "optimal" bid.

Goal Representation

The non-adaptive bidding strategies, *Truth-Telling* and *ZI-C*, do not define a scoring function (a goal) and do not adapt based on the rewards received from the actions generated. All other bidding strategies implement a scoring function, which is used to update the agent's knowledge base from the rewards received and the time they are received. Applied mostly in financial market settings, these strategies aim to maximize the agent's profits, which is the result of the agent's valuations less the

clearing prices. *Evolutionary search* and *Straightforward Bidding* do not aggregate the rewards received from each action; their aim when calculating the best response strategy is profit maximization. The RE bidding strategy reinforces the discounted rewards received from agent' actions in its knowledge base (propensity value for each of the actions), which is used in the bid generation phase to select the most successful action. The P Strategy uses RL techniques to calibrate the probability transition parameters from the rewards received. Furthermore, the P Strategy computes the utility values for each of the possible actions (bids) and selects the action with the highest expected utility. The ZIP and GD strategies optimize immediate profits when generating their bids. The GDX strategy optimizes the cumulative long-term discounted profits of the actions using dynamic programming. AA optimizes long-term profits using ZIP-based (short-term) reactions for on changing market conditions. Agents in the Schwartzman and Wellman (2009) model receive immediate rewards, which are used in their RL model to update the state information.

Adaptive Bidding in Imperfect Markets

Almost all of the bidding strategies in Table 3.1 are designed to work in perfect information markets (all agents' bids and prices are common information). The *Truth-Telling* and *ZI-C* strategies behave the same in perfect and imperfect information markets. *Straightforward Bidding* can calculate a best response only when information about the other bidders is available. The decision making functions of the remaining strategies are designed to work with perfect information, however, each of these algorithmic bidding strategies partially implement RL techniques in some of their decision steps, e.g., when calibrating variables or updating cumulative rewards. Such RL techniques can also work (in the long-term) with only partially available information. Nevertheless, these bidding strategies are designed to work with perfect information and evaluated in auction settings where all agents are informed about other agents' actions. Therefore, their properties and outcome efficiency in the imperfect information case might be completely different or difficult to explain.

Bidding in a Market-Based Scheduling Domain

In Table 3.1 only *Straightforward Bidding* is explicitly designed for a market-based scheduling scenario. *Straightforward Bidding* is defined for a scenario for allocating CPU slots in a simultaneous ascending auction with perfect information about other agents' actions. However, the auction mechanism and the bidding strategies discussed in Reeves et al. (2005) remain at a higher abstraction level by applying classic economic assumptions, idealistic (perfect) information and strategic (best response) models. All other bidding strategies are evaluated in financial market scenarios, i.e.,

\widehat{CDA} with specific time constraints. In a market-based scheduling scenario, the market mechanisms are assumed to run infinitely, i.e., non-constrained continuous and autonomous allocation of applications to idle computing services with “intelligent” agents and bidding strategies in *CDA* market types. Moreover, none of the strategies of past and current research discussed here provide an explicit definition of communication protocols (see Chapter 5) for agent interactions in a market-based scheduling domain, i.e., a well-defined procedure for mapping the endogenous and exogenous variables (strategic and technical) of bidding strategies, such as state, action, information, reward model and a well-defined message type. In real market-based scheduling scenarios agents need to communicate according to well-defined and commonly accepted communication protocols, which explicitly define the type of information exchanged and rules of the market mechanism used. Moreover, the target domain of this work requires a definition of bidding strategies that models economic and technical attributes in their state, action, information and reward representations. Another important limitation of the current research on market-based scheduling domains is the focus on profit maximization. However, according to their key performance indicators, consumers and providers of computing services will adopt more complicated scoring functions, e.g., min. completion time and payments, max. utilization, max. reliability, min. penalty etc. (Wilkes, 2008; Auyoung et al., 2009; Brynjolfsson et al., 2010).

3.4 Q-Strategy: A Bidding Strategy for Automated Provisioning and Purchasing of Computing Services

3.4.1 Why Reinforcement Learning-Based Bidding?

Today’s software applications and computing services are highly heterogeneous. Owners of software applications have different requirements on computing infrastructures for deploying and executing their applications; and providers maintain computing infrastructures based on hardware components from different manufacturers.⁶ Therefore, the strategies that consumers and providers use have to be able to handle uncertainties when computing services are purchased or provided through the market. Unlike supervised and unsupervised learning, where agents learn from past input and

⁶Symantec (2008) reported that, on average, companies work with more than one thousand applications; almost all of the computing centers on the Top500-List (<http://www.top500.org>) are built with different hardware specifications.

output data, *reinforcement learning* (RL) enables the learning from local and feedback data, which is received from the environment (Sutton and Barto, 1998; Tesauro, 2007). RL incorporates this feedback according to two intuitive effects from the psychology and control theory – *Law of Effect* and *Power Law of Practice*. The *Law of Effect* postulates that agents are more likely to select the action that achieved the highest average performance in the past. The *Power Law of Practice* postulates that learning curves tend to be steep at the beginning and flatten out over time. Moreover, the feedback collected can be used immediately in the generation of new actions. Therefore, RL-based mechanisms are applicable in interactive (online) settings, where agents can explore the environment and adopt “better” actions in order to improve their rewards over time.

In environments with imperfect information, a *model-free learning* mechanism is more suitable than a *model-based learning* mechanism since the endogenous variables are actual and well known to the owners, whereas the exogenous variables might be incomplete, incorrect or obsolete.

General Reinforcement Learning models, in particularly Q-Learning, define the general framework of an interactive and adaptive environment of agents’ states, actions, rewards and transition policies. Concrete specifications, realizations and evaluations for (real-world) application scenarios are completely missing within the Q-Learning framework. In the next section, the theory behind Q-Learning will be briefly presented. In the subsequent section, a specification, realization and complexity analysis of a novel bidding strategy for the market-based scheduling domain, Q-Strategy, is presented.

3.4.2 The General Q-Learning Framework

The theory of reinforcement learning was developed by Bellman in the 1950s, who formulated the Bellman equation for solving the infinite horizon problems of control theory (Sutton and Barto, 1998). The Bellman equation uses a dynamic programming approach to accumulate discounted rewards for observed state-action pairs in order to “learn” the optimal actions from recurring experiences in an infinite game. In static environments of single-agent (self-play) learning with known states, actions and rewards, it has been shown that Q-Learning converges to optimal values over time when all actions are infinitely and repeatedly explored (Watkins and Dayan, 1992b; Tsitsiklis, 1994; Bowling, 2000). There is also research on the multi-agent case, but there is no clear proof that Q-Learning converges in cases of cooperative and perfect information or of what happens in non-cooperative imperfect information

games (Shoham et al., 2004, 2007; Stone, 2007b).

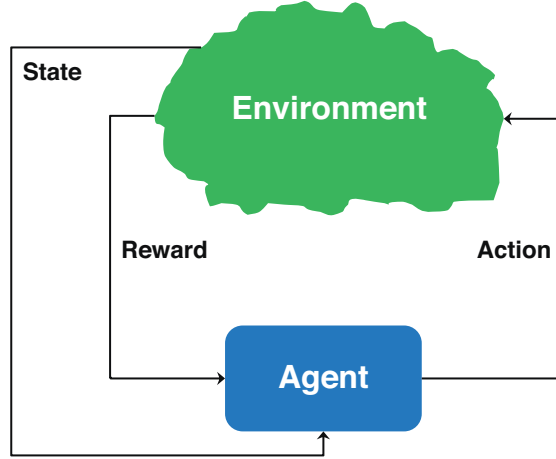


Figure 3.1: Standard reinforcement learning interaction loop (adopted from Kaelbling et al. (1996))

The general interaction loop of the Q-Learning framework is depicted in Figure 3.1:

- *Step 1: State Observation.* An agent i receives as input an observation for a state $\omega_{i,k}$ of the environment;
- *Step 2: Action Selection.* Based on the transition probability function ρ , the agent selects an action $a_{i,k}$ for state $\omega_{i,k}$, which is executed in the environment;
- *Step 3: Reward Calculation.* The agent receives a reward $r_{i,k}$, which is calculated from a reward function R ;
- *Step 4: Value Update.* The reward $r_{i,k}$ for the state-action pair $\langle \omega_{i,k}, a_{i,k} \rangle$ is aggregated according to the *Q-Value update rule* in order to reflect the new observations;
- *Step 5: State Transition.* A state transition policy identifies the next state $\omega_{i,k+1}$ to be observed, which moves the algorithm to *Step 1*.

For each observed state-action pairs, the *Q-Value update rule* cumulates the received reward in a so-called *Q-Value*. The $n + 1$ experience (*Q-Value*) of the state-action pair $\langle \omega_{i,k}, a_{i,k} \rangle$ is updated according to the following rule:

$$Q^{n+1}(\omega_{i,k}, a_{i,k}) = Q^n(\omega_{i,k}, a_{i,k}) + \beta_{\omega_{i,k}} [r_{i,k} + \gamma \max_a Q(\omega_{i,k}, A_i) - Q^n(\omega_{i,k}, a_{i,k})] \quad (3.10)$$

$Q^n(\omega_{i,k}, a_{i,k})$ is the current *Q-Value* of a state-action pair, which is updated into $Q^{n+1}(\omega_{i,k}, a_{i,k})$ by discounting the received reward $r_{i,k}$ with the maximum expected value for future rewards (with respect to the best action $\arg \max_a(A_i)$, A_i are the explored actions for state $\omega_{i,k}$) and the old Q-Value of the target state-action pair. $\beta_{\omega_{i,k}} \in [0, 1]$ is the learning rate, which determines how much weight is given to newly observed rewards. A high value of $\beta_{\omega_{i,k}}$ results in high importance being assigned to the current observed reward, while a low value leads to the use of small steps to update the *Q-Value*-function. $\beta_{\omega_{i,k}} = 0$ means that no learning will occur at all. The discounting factor γ defines how much expected future rewards affect current decisions. A low value ($\gamma \rightarrow 0$) implies greater attention to immediate rewards. A higher value ($\gamma \rightarrow 1$) implies orientation towards future rewards, whereby agents may be willing to trade short-term loss for long-term gain.

Typical values in the research for the learning rate $\beta_{\omega_{i,k}}$ have the range $[0.01, 0.3]$ and for γ the range $[0.1, 0.9]$. The estimation of these parameters constitutes a trade-off between the ability to consider the dynamics of the environment and thus the reaction to changing conditions. Whiteson and Stone (2006) proposed an evolutionary search of the Q-Learning parameters for the mountain car and server application scheduling scenarios. Sun and Peterson (1999) varied the learning rate through a heuristic policy. In Even-Dar et al. (2003) and Even-Dar and Mansour (2004) performed stochastic searches for estimating the Q-Learning parameters in stationary settings.

Another trade-off is the choosing between exploration and exploitation mode, i.e., to explore new action values or to use the cumulated knowledge for selecting the current “best” action for a given state $\omega_{i,k}$. The exploration and exploitation trade-off, also called the *epsilon-greedy* policy (ϵ), has been theoretically studied in so-called multi-armed bandit problems (MAB). MAB is part of reinforcement learning theory and describes a solution for iterative calibration (trial and error) of states to actions (control theory), i.e., for estimating variables for a given state through repeated execution of the actions and incorporation of the received feedbacks. The literature uses different values and value ranges for $\epsilon \in [0.01, 0.5]$ in order to balance exploring new knowledge and exploiting already existing knowledge for the action selection. In the evaluation part of this work, the $\beta_{\omega_{i,k}}$, γ and ϵ variables are estimated based on the suggested value ranges and a sensitivity analysis, in which each of the parameter is varied and the parameters combination with the highest performance (outcome) is selected.

Q-Learning is a well-explored general framework in learning theory. However, as such, it defines abstract concepts and their relations, but the definition of concrete

learning models for specific environments (e.g., soccer robots, online human-machine games like poker and chess, autonomous car driving), their concrete specification, realization and evaluation is a design and research problem (Stone, 2007a). The next section presents a novel bidding strategy for the market-based scheduling environment called the *Q-Strategy*, which utilizes the Q-Learning concepts, and expands on them with concrete and detailed specification and realization details. The *Q-Strategy* is evaluated in Chapter 6.

3.4.3 The Q-Strategy Model

3.4.3.1 Automating the Bidding Processes

Most of the bidding strategies in Table 3.1 do not include detailed evidence of their specification, implementation and complexity analysis. However, many of them apply evolutionary search techniques for calculating best response actions based on market information of other agents' bids and clearing prices. Such techniques require exponential efforts with respect to the number of agents and their actions.

The aim of the *Q-Strategy* model is the automation of consumers' and providers' bidding processes in markets for trading computing services (Section 2.3.2) and for two real case studies for batch and interactive applications (Section 2.4.1). Consumers and providers use the well-defined interfaces of the *Q-Strategy* to configure their states, goals and dynamic parameters (exploration rate, learning rate and discount factor). The *Q-Strategy* is initialized into bidding agents (Section 4), which start the execution of the bidding processes and manage the communication with the environment (Section 5). Therefore, the *Q-Strategy* contributes to the research on autonomic computing by automating the decision making processes of consumers and providers when participating in markets for computing services.

3.4.3.2 State Representation

Consumers use heterogeneous applications with heterogeneous requirements on the computing services (see scenarios in Section 2.4.1). Providers maintain heterogeneous computing infrastructures from different hardware manufacturers, which are designed for various use cases and application scenarios. Therefore, the design decision of a *Q-Strategy* state definition should reflect this in supporting bidding processes for heterogeneous applications and computing service descriptions. The innovative approach of Q-Strategy's state is its representation as a *multi-armed state machine* with a *multi-armed action learner*. This approach not only allows the action space

for a given state to be explored, but also the expression of policies to create new states.

For example, a consumer (company) has a large log data of last month's sales and uses the TXTDemand application (Section 2.4.1.1) to analyze the sales and calculate the replenishment strategy for the following month. The consumer estimates that his technical requirements will be 2 CPUs with at least 2 GHz, 4 GB of dual-channel memory and 30 GB disk space. Moreover, the log data should be analyzed as quickly as possible (estimation max. of 8 hours) for not more than 5 monetary units (\$). The bidding processes are executed with the preferred bidding strategy and the TXTDemand application is allocated for 3 monetary units on a machine, which satisfies these requirements and needs 7 hours to complete the job. The next time, the consumer decides to increase the number of CPUs to 4 and the memory requirements to a three-channel 4GB memory. The reward was that the applications were finished in 5 hours and within the 5 monetary units constraint.

Such manual optimizations can be automated through state transition policies, which are supported per design in *Q-Strategy's* state definition. Moreover, the definition of *Q-Strategy's* state does not capture application specific data like deployment, runtime and termination instructions, but the technical requirements of the applications (providers' technical descriptions of their computing services) for the computing services with the economic constraints – duration, valuation and goal. Therefore, the same consumer may want to execute a ray-tracing application for her or his collection of holiday pictures and with the same technical requirements and economic preferences for a computing service at a later time. Here, the *Q-Strategy* will exploit the knowledge already collected from past bid generation processes with the same state description. The applied technique is called *Learned Abstractions*, i.e., the grouping of similar states according to a well-defined “similarity rule,” i.e., similar technical descriptions (key performance indicators) and economic preferences.

The *Q-Strategy* state for an agent i is represented through the 6-tuple

$$\Omega_i = \{\Theta_i, c_i, v_i, U_i, \Phi_i, \rho_\omega\}:$$

- *Technical Description.* Θ_i defines technical attributes for the consumer's requirements and the provider's technical description like CPU, memory, storage and network bandwidth;
- *Duration.* c_i is an (upper-bound) estimation in time units for the total time a computing service is demanded or available on the market;

- *Valuation.* v_i is the maximum monetary unit a consumer is willing to pay per unit of time for the application's execution on a target computing service. For providers, it is the minimum requested price for the computing service per unit of time;
- *Scoring Policy.* U_i is the scoring function that consumers or providers adopt in order to achieve their goals in the bidding process (see *Goal Representation*);
- *State Abstraction Policy.* $\Phi_i : \Omega_i \times \Omega_j \rightarrow \Omega_i$ is the policy, which is defined by consumers and providers. It clusters their requests of similar states $\Omega_i \approx \Omega_j$ with $\Theta_i = \Theta_j$, $c_i = c_j$, $v_i = v_j$ and $U_i = U_j$ to one single unique state description Ω_i ;
- *State Transition Policy.* The state transition function $\rho_\omega : \Omega_i \times \xi \rightarrow \tilde{\Omega}_i$ defines the policy ξ , which adapts the attributes of the technical description in order to explore new states for the specified goal.

The technique for clustering similar states, *State Abstraction* is especially applicable to applications, which are repeatedly executed with technical requirements and input data that does not change often.

3.4.3.3 Action Representation and Adaptive Bidding

Figure 3.2 shows an example of Q-Strategy's concepts. On the left, it shows the different states of the agent. In time t , an agent i receives a "request for bid," and based on the information in \mathcal{P}_i^t , the *state arm* looks for the state,

$\omega_{i,2} = \{\{4CPU, 2GB Memory, 10GB Disk\}, 5h, 30\$, "minCompletionTimeAndPayment"\}$, which matches the description in \mathcal{P}_i^t : $\Theta_i^t, c_i^t, v_i^t, U_i^t \in \mathcal{P}_i^t$ (see the definition of \mathcal{P}_i^t in Section 5.3.1).

For $\omega_{i,2}$, the agent has explored m actions so far. Based on the action selection policy, the *action arm* returns the action, which is to be executed in the environment. Here, the "best action" with the current highest *Q-Value* is $a_{i,2}$ with $Q(\omega_{i,2}, a_{i,2}) = 67$. This example also shows the change of the *Q-Value* over time, based on the received reward from the action execution. Here, $a_{i,2}$ was executed twice with rewards of 64 and 67. The *Q-Value* is the cumulative reward, which is calculated with the *Q-Value update rule*. For example, the last *Q-Value* of $a_{i,2}$ is calculated with $Q^1(\omega_{i,2}, a_{i,2}) = 64$, a current expected value of $\max_a Q(\omega_{i,k}, A_i^k) = 64$, $\beta_{\omega_{i,2}} = 0.1$ and $\gamma = 0.9$:

$$Q^2(\omega_{i,2}, a_{i,2}) = 64 + 0.1(38 + 0.9 \cdot 64 - 64) = 67 \text{ (rounded)} \quad (3.11)$$

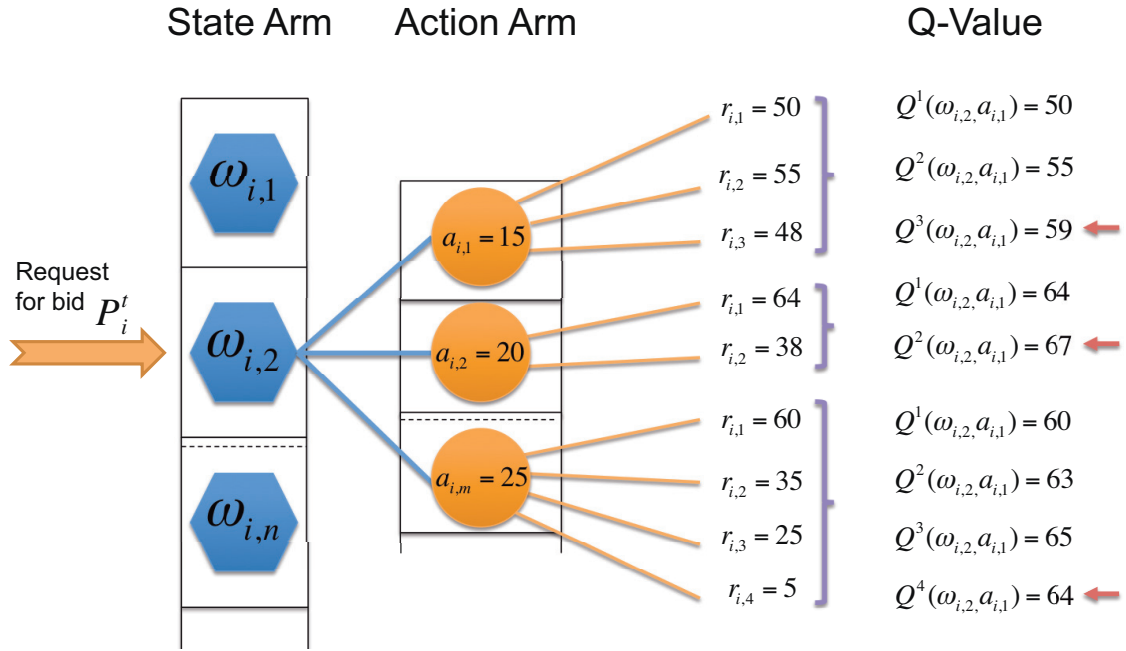


Figure 3.2: Example of Q-Strategy's multi-states with multi-action arms

The adaptive bidding processes in the *Q-Strategy* are executed in two main phases – the *Exploration* and *Exploitation* phase, according to the ϵ -greedy action selection policy (Kaelbling et al., 1996).

Exploration Phase. The exploration phase allows users to learn from and adapt to their environment based on new or randomly selected actions and the rewards associated with these actions. In this phase, for a given state $\omega_{i,k}$ the *Q-Strategy* uniformly generates an action $a_{i,k}$ with upper and lower bounds from the valuation of the state $v_i \in \omega_{i,k}$, $U(zv_i, v_i)$ and factor z – for the consumer $z \in [0, 1]$ and for the provider $z \in [1, \infty]$. The motivation of this phase is that market environments are dynamic over time and agents will continuously experience different rewards from each of their actions. However, the continuous accumulation of these experiences will allow better decisions in the exploitation phase.

Exploitation Phase. The exploitation phase selects the action $a_{i,k}$ in a given state $\omega_{i,k}$, which achieved the highest cumulative reward over time. The cumulative reward $Q(\omega_{i,k}, a_{i,k})$ implements the *Law of Effect* and *Power Law of Practice* policies and is calculated with the *Q-Value update rule* for each executed action $a_{i,k}$ in $\omega_{i,k}$. The information about the states, associated actions and their cumulative rewards are stored in the agent's knowledge base in a so-called *Q-Table*.

Equation 3.12 summarizes the decision function between the exploration and exploitation states:

$$a_{i,k} = \begin{cases} \arg \max_a (Q(\omega_{i,k}, A_i)) & \text{with probability } 1 - \epsilon \\ U(zv_i, v_i) & \text{with probability } \epsilon \end{cases} \quad (3.12)$$

With a probability of ϵ , the *Q-Strategy* selects an action within the *Exploration Phase* and with a probability of $1 - \epsilon$ it selects an action within the *Exploitation Phase*, for each incoming “request for bid.” In cases with missing information about a given state, the $\epsilon \in [0, 1]$ value can be set to select the *Exploration Phase* with a higher probability, e.g., $\epsilon = 0.7$, and reduced with time to $\epsilon = 0.1$ in order to exploit and refine the initial knowledge through the *Exploitation Phase*. The z parameter is predefined and set by the consumers and providers. It is also possible to adapt it over time, however, this extension will be a part of future research.

In summary, the *Action Representation* model allows continuous adaption over time and to the market conditions. Continuous adaption is implemented with the visit of the *Exploration Phase* with the probability of ϵ , which remains constant over time.

3.4.3.4 Goal Representation and Q-Value Update

The *Q-Strategy* does not predefine the scoring function U_i for all states, but provides a built-in flexibility of the *State Representation* to allow consumers and providers to specify their own scoring functions, which are part of the matched state $U_i \in \omega_{i,k}$ for their “request for bid.” When there is no available state that matches the “request for bid,” a new unique state is created in *Q-Strategy*’s *multi-armed state machine*. Therefore, the scoring functions are decoupled from the *Q-Strategy* implementation and provided as external rules, which are called up from the *Q-Strategy* for the reward calculation. The different scoring function identities are provided as a list to the consumers and providers, from which they select their preferred scoring function for the traded transaction object. The *Q-Value* is updated after each action execution $a_{i,k}$ for the matched state $\omega_{i,k}$, according to the specified scoring function in U_i , which is used as the reward calculation function, $R \equiv U_i$, in the *Q-Value update rule*.

For example, in an application case, consumers can select a “minimize completion time” scoring function; in other application cases they would select “minimize payment” or both at the same time, “minimize completion time and payment.” Analogously, providers of computing services would select “maximize profit,” “maximize utilization” or both, “maximize profit and utilization.” The evaluation of existing

scoring functions is not a part of this research, but the application of existing ones is part of it. In this work, two scoring functions are selected as relevant for the evaluation: consumers of computing services that “minimize completion time and payment” as stated in Heydenreich et al. (2010), and providers that are classically defined as profit (the difference between valuation and clearing price) maximizers. Selection of the profit maximizing scoring function allows the agent outcomes to be compared with state-of-the-art works for evaluating bidding strategies. Selecting of the “minimize completion time and payment” scoring function allows for a more realistic analysis of bidding strategies in the market-based scheduling domain, which are largely unexplored in the current related research.

3.4.3.5 Realization of the Q-Strategy for a Market-Based Scheduling Domain

Figure 3.3 presents the realization architecture of the *Q-Strategy* in the *BidGenerator* framework (cf. Figure 4.2 in Section 4.4). States and actions are implemented as inner classes of the *Q-Strategy* class.

The *State* class implements the mapping of “request for bids,” implemented as *PrivateMessages* (Section 5.3.1), to the states of *Q-Strategy*’s *multi-armed state machine*. The *Action* class describes the mapping between an action and its cumulative reward. The *Q-Strategy* class implements the *generateBid* method (Algorithm 3.4.1), which generates a bid with respect to the decision of an exploration or exploitation phase. The result of the *generateBid* is the created *PublicMessage* (Section 5.3.3), which is submitted to the market. The *Q-Strategy* class method *updateCalculateScore* receives status information, *StateMessage* (Section 5.3.2), on the transaction object (application or computing service) execution and calculates the achieved reward based on the selected scoring function U . Additional information like *clearing price* is extracted from the market match-message (see *MarketMessage* in Section 5.3.4), which was received from the market on successful allocation (Section 5.4) of the consumer and provider bids. At the end, the *updateCalculateScore* invokes the *newState* method of *QLearner* in order to accumulate the received reward from the action execution of the target state with the *Q-Value update rule*.

The states to actions cumulative reward information is stored in the *Q-Table*. The *Q-Table* is implemented in memory within the *QLearning* data management structure, but can be made persistent in a database with little effort.

The *QLearner* class implements the maintenance of states to actions, the update of the cumulative rewards (*newState*) and the retrieval of the *bestAction*. Originally, the

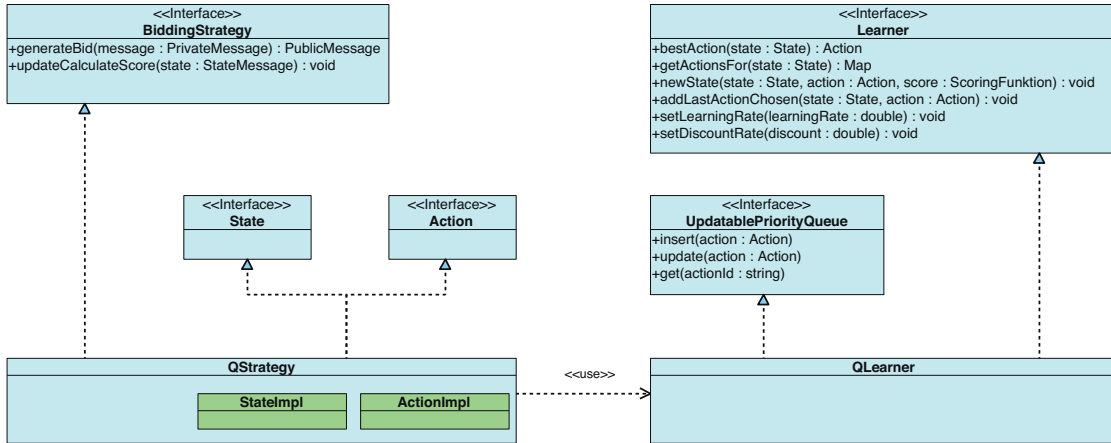


Figure 3.3: Q-Strategy’s implementation in BidGenerator

QLearner implementation was inspired by the JASA framework⁷, but in its current version the *BidGenerator*’s *QLearner* is a significant extension to JASA’s implementation since now it i) supports *States* and *Actions* as objects rather than integer arrays and ii) implements computationally efficient management of *Actions* with a priority queue technique. The *UpdatablePriorityQueue* interface specifies general methods of managing *Actions* in an efficient and sorted *Heap* data structure.

Algorithm 3.4.1 describes the implementation of the *generateBid* method. *generateBid* receives the “request for bid” message in the form of a *PrivateMessage*. It initializes the *valuation*, the provider’s or consumer’s *z*-factor and the *state* information from the *PrivateMessage*. The selection between the *Exploration* and *Exploitation* phase is determined by the ϵ parameter, which is compared to an automatically drawn parameter from a uniform distribution. The *Exploration* phase generates a uniform bid as specified, whose value is bound within the consumer’s or provider’s valuation and their *z*-factor scaled valuation. In the *Exploitation* phase, the bid is the current *best action* in the *Q-Table* for the target state. As a last state, the bid is incorporated into the *PublicMessage*, which is a subset and transformed from the *PrivateMessage*.

⁷<http://sourceforge.net/projects/jasa>.

Algorithm 3.4.1: Q-STRATEGY: BID GENERATION RULE(*privateMessage*)

comment: Generates a *bid* according to the current strategy phase:

* Exploration phase generates random bids

* Exploitation phase exploits the state-action-reward history

procedure GENERATEBID(*privateMessage*)

valuation \leftarrow *privateMessage.getValuation()*

if *privateMessage.isConsumerBid()*

then $\{z \leftarrow$ *privateMessage.getConsumerZ()*

else if *privateMessage.isProviderBid()*

then $\{z \in$ *privateMessage.getProviderZ()*

if $\epsilon < \text{Stochastic.uniform}(0, 1)$

then **comment:** Explore:

$bid \leftarrow \text{Stochastic.uniform}(z \cdot valuation, valuation)$

comment: Exploit:

$state \leftarrow \text{State.getState}(\text{privateMessage})$

$action \leftarrow \text{QLearner.bestAction}(state)$

if $action \neq nil$

then $\{bid \leftarrow action.getBid()$

else if

then **comment:** Q-Table is empty for *privateMessage* def.

$bid \leftarrow \text{Stochastic.uniform}(z \cdot valuation, valuation)$

return (*bid*)

3.4.3.6 Complexity Analysis

The bidding strategies discussed in Table 3.1 do not provide any detailed evidence of the complexity of their implementations. Bidding strategies like *Straightforward Bidding*, *Evolutionary Search*, *P-Strategy* and *Schwartzman-Wellman* perform evolutionary search for best response strategies based on the available public information of other agents' actions – bids and clearing prices. Evolutionary search techniques solve complex problems, but suffer from exponential computational complexity due to the number of agents and their actions. Moreover, reinforcement learning research focuses on theoretical concepts of learning, but does not discuss possible implementation solution concepts and their complexity.

Papadimitriou and Tsitsiklis (1987) proved that Markov Decision Processes are *P-complete* and thus solvable in polynomial time:

Theorem 3.4.1 (Complexity of a Markov Decision Process). *The Markov Decision Process problem is P-complete in all three cases, finite horizon, discounted and average cost.*

Therefore, the theoretical computational complexity of the *Q-Strategy* concept is *P-complete*, and the complexity of practical implementation is discussed as follows. Currently, the *Q-Strategy* is provided with an in memory implementation. The *multi-armed state machine* for the management of *Q-Strategy's states* descriptions is implemented with *Hashtable*, which includes a $O(1)$ complexity for lookup, add, update and delete states. Rare events like resizing the *Hashtable* add an additional short-time complexity of $O(n)$ with the number of n states that can be significantly reduced in average when selecting an appropriate initial size of the table and a table size improvement policy.

The actions of each state are organized in a *max-Heap* data structure, implemented with the *UpdatablePriorityQueue* class. A *max-Heap* data structure is an efficient implementation of a priority queue, where the top element has the highest value and all other elements are ordered in a binary tree form, where the last element has the lowest value (Sedgewick and Wayne, 2010). Moreover, the *UpdatablePriorityQueue* implementation allows the cumulative reward values of the associated actions to be updated dynamically. The dynamic update refers to the fact that the updated cumulative rewards of the associated actions have to be reordered in cases where consistency of the *max-Heap* data structure is violated. This implementation allows a very efficient search for the *best action* since this action has the highest cumulative reward, therefore it is the first element in the *max-Heap* and accessible in $O(1)$. Moreover, during the *Exploitation Phase* the *UpdatablePriorityQueue* will deliver and update actions close to the top element with an average lookup and update complexity of $O(1)$. During the *Exploration Phase*, the lookup and update operations can have $O(\log n)$ complexity in the worst case scenario.

3.4.3.7 Critical View

The aim of the *Q-Strategy* is to enable agents to perform quasi real-time decisions on bid generation processes for their applications or computing services, which is an important issue for the emerging markets in computing services (Amazon, 2010a; TheCloudMarket, 2010). The *Q-Strategy* provides a theoretical and architectural solution concept, which is applicable, but not limited to a market-based scheduling domain. A further advantage of the *Q-Strategy* is that it can work without public information, which is not the case for bidding strategies like ZIP and GD that only work when public information of other agents' actions is available. Therefore, *Q-Strategy* is able to adapt to local information with regard to states, actions and their received rewards. The application scenario of the *Q-Strategy* is motivated by the fact

that the environment is not fully transparent and agents do not share their private information with each other. Moreover, the *Q-Strategy* allows actions and their cumulative rewards to be learned over time for different states and scoring functions. The *Q-Strategy* is designed to be flexible and adaptable in dynamic environments, as well as implemented to be computationally tractable with *P-complete* complexity.

However, in the worst case scenario, reinforcement learning techniques may produce outcomes with higher opportunity costs (i.e., lower rewards) at the beginning, but adapt to optimal decisions over time (Watkins and Dayan, 1992a; Tesauro, 2007). The convergence of *Q-Learning* to optimal values over time is shown theoretically only in stationary settings of one agent or in cooperative games (Bowling, 2000). Reinforcement learning algorithms also have to implement efficient trade-offs between exploration and exploitation in order to i) learn from and exploit the experience, but also to ii) preserve the learning processes in changing environments. Therefore, *Q-Strategy* might need to observe a high number of different state-action pairs in order to derive the optimal states. Such a number could probably be estimated in stationary environments, but not in dynamic ones. Moreover, the received reward values for the actions may be noisy, due to the fluctuations in supply and demand over time and bidding strategies of other agents. The current research shows little evidence of more realistic and dynamic scenarios in settings with multiple heterogeneous agents, in which different types of algorithmic bidding strategies are applied (Shoham et al., 2007; Tesauro and Bredin, 2002).

3.5 Summary

This chapter introduced the general design desiderata for developing bidding strategies, as well as existing bidding strategies for games with perfect and imperfect information. A set of bidding strategies has been extensively evaluated to determine if they qualify as candidates for a market-based scheduling domain. There is no bidding strategy available, which satisfies the design desiderata, in particular, for the target domain of this work. Therefore, this chapter presents the design and realization of the novel bidding strategy, *Q-Strategy*, for the automation of, but not limited to bidding processes in markets for computing services.

Q-Learning is a generic learning framework, which defines the general relations of states, actions, rewards and state transitions. In contrast, the *Q-Strategy* contributes to state-of-the-art learning-based bidding strategies by i) specifying the state, actions, goal representations and their transitions explicitly. *Q-Strategy's state* definition al-

lows heterogeneous trading objects with different scoring functions to be represented, which are maximized through the *actions* of the *Q-Strategy*. In summary, ii) the *Q-Strategy* solves a multi-armed bandit problem for trading heterogeneous transaction objects. Similar “requests for bids” are iii) clustered into unique states, which enable faster collection of experiences for recurring executions of applications and computing services. The realization part shows the iv) implementation of *Q-Strategy* concepts within a framework for automated bidding called *BidGenerator* (Section 4.4). The *Q-Strategy* is integrated in the communication protocol (Section 5.3) of the *BidGenerator* framework and a system for market-based scheduling for exchanging private, public and market information (cf. bidding scenario in Section 2.4.4). Finally, v) the *Q-Strategy* is designed to act on and fully adapt to local information and past experiences in dynamic market environments of imperfect information. Steps i) to v) are properties of the *Q-Strategy* model. None of the steps are part of the general *Q-Learning* framework. Moreover, *Q-Strategy* is provided with a complexity analysis of its realization, which confirms its practicability in real application scenarios and for the investigated domain.

This chapter presented the economic design of an existing and novel bidding strategy. The next chapter introduces the *BidGenerator* framework for realizing bidding strategies together with software agents. The latter manages the communication processes with applications, computing service managers and the target market mechanism according to a well-defined communication protocol (Chapter 5).

Chapter 4

Architectural Design of a Framework for Automated Bidding

THE computing literature has mainly focused on the “brawn” – development of technical infrastructures and related tools for managing and providing reliable, secure and distributed computing resources – and less on the “brain” – for automating their provisioning and purchasing processes efficiently (Foster et al., 2004). Moreover, mechanisms for market-based scheduling of computing services have been developed, but there has been less effort put into the development and evaluation of bidding agents and strategies for the market-based scheduling domain (Broberg et al., 2008; Chevaleyre et al., 2006; Lubin et al., 2009).

This chapter presents a novel framework for automated bidding called *BidGenerator*, in which agents and bidding strategies are realized and coexist as independent, configurable modules. *BidGenerator* is the solution concept, which aims to answer Research Question 2 (*Design of a Framework for Automated Bidding*) in Section 1.2. The chapter starts with a definition of what a bidding agent is (Section 4.1). Design desiderata for developing bidding agents from a technical perspective are derived (Section 4.2) and existing agent frameworks according to the desiderata evaluated (Section 4.3). Section 4.4 presents the architecture of the *BidGenerator* framework and discusses the fulfillment of the desiderata. Section 4.5 is a summary of the chapter.

4.1 What is a Bidding Agent?

In contrast to the economic definition of a bidding strategy (Section 3.1), the bidding agent is a software entity, which has a certain and well-defined functionality and is able to act automatically on behalf of its owner. More formally, Jennings (2001) defines an agent as:

Definition 4.1.1 (Agent). *Agents are “clearly identifiable **problem-solving entities** with **well-defined boundaries and interfaces**; **situated (embedded) in a***

*particular environment over which they have **partial control** and observability – they **receive inputs** related to the state of their environment through sensors and they **act** on the environment through effectors; designed to **fulfill a specific role** – they have particular **objectives** to achieve; **autonomous** – they have control both over their **internal state** and over their **own behavior**; capable of exhibiting flexible problem-solving behavior in pursuit of their **design objectives** – being both **reactive** (able to respond in a timely fashion to changes that occur in their environment) and **proactive** (able to opportunistically adopt goals and take the initiative).”*

This definition gives a clear description of the spectrum of a software agent with respect to the specific goals that they are designed to fulfill in a specific domain. The differences between a software application and a software agent are their respective properties (Wooldridge and Jennings, 1995):

- *Autonomy.* Agents operate without the direct intervention of humans and have control over their actions and internal state;
- *Social Ability.* Agents interact with other agents (including institutions, such as running markets) in the environment via well-defined communication protocols;
- *Reactivity.* Agents perceive their environment and respond in a timely fashion to changes that occur in it;
- *Proactive.* Agents do not simply react to their environment, they are able to exhibit goal-directed behavior by initiating actions.

Both, Wooldridge and Jennings (1995) as well as Nwana (1996) give clear overview of agent design, agent architectures and agent languages, which are principles that are also relevant today, and more recent papers ascribe to the approaches and taxonomies introduced by these authors. A more recent overview and comparison of existing frameworks is provided in Section 4.3.

Nwana (1996) defined a taxonomy of agent types – *Collaborative agents, Interface agents, Mobile agents, Information/Internet agents, Reactive agents* and *Hybrid agents*. In the context of this work, the developed bidding agent framework can be assigned to the type of *Hybrid Agents* containing interface, information gathering and collaboration facilities. The *Interface Agent* (IA) type refers to software agents, which act and learn autonomously on behalf of their owners. Based on the state of the environment and given goals, the agents execute delegated tasks autonomously

by taking appropriate actions in the environment. IAs assist their owners in reaching a goal and are characterized by their iterative (proactive) and adaptive behavior. *Information (Gathering) Agents* (IGA) implement algorithms for the periodic collection and aggregation of data. In contrast to Web Crawlers and within the context of market-based scheduling, IGAs implement facilities to collect and aggregate market information (volatility, offered computing service types, past and current prices, market rules, etc.) from various sources and markets. *Collaboration Agents* (CA) focus on facilities for message exchange (negotiation or bidding) between other agents in the environment for reaching a given goal with respect to other agents' desires and actions. CAs implement social facilities in terms of communication protocols for negotiations or bidding, as well as adapt to other agents' responses.

The aim of the *Hybrid Agent* type is to reduce complexity by introducing the modularization of specific types of agent facilities – automatic acting on behalf of the owners based on their preferences, automatic information gathering and aggregation, as well as proactive and collaborative facilities by negotiation and bidding.

In an environment for market-based scheduling, where institutions, consumers and providers are independent and distributed entities, *Hybrid Agents* seem to offer the correct type of characteristics to implement an autonomous system. In computing service markets, *Hybrid Agent* architectures implement the “brain” and has the potential for effective, flexible and decentralized decision making capabilities. Such agents need “a robust distributed computing platform that allows them to discover, acquire, federate, and manage the capabilities necessary to execute their decisions” (Foster et al., 2004).

In the context of market-based scheduling, a bidding agent is defined as follows:

Definition 4.1.2 (Bidding Agent). *A Bidding Agent is a software entity, which provides well-defined **interfaces for instantiating bidding strategies, interfaces for information gathering and aggregation, as well as interfaces for proactive and reactive communication** with target market mechanisms or other agents. Based on the **inputs** received, the bidding agent maintains an internal state of the owner's tasks, and **acts** and **reacts** with the environment on the owner's behalf by performing decision making actions.*

4.2 Design Desiderata

The following design desiderata for agent frameworks in a market-based scheduling domain are derived from the technical challenges and application requirements pre-

sented in Section 2.4, as well as from current research.

Desideratum 1 <Decoupling of the Market Platform and Bidding Agents>

The market platform and BidGenerator must be separated into independent components, the first, operated and maintained by an auctioneer, the second, by their owners – consumers and providers.

A realistic system for market-based scheduling is a decentralized, multi-agent system, where the market, consumers and providers are autonomous entities. The application scenarios in Section 2.4 postulate that each of the consumers and providers aim to use their own bidding agents locally by maintaining control of their realizations and bidding strategy selections. The agents are decentralized and the communication is performed asynchronously. Furthermore, there is no central entity, which controls all the agents, i.e., agents can join and leave the system at any time. Therefore, the market platform and bidding agent framework must be separated into independent components, the first, operated and maintained by an auctioneer, the second, by their owners (Sycara, 1998; Tesauro et al., 2004; Chevaleyre et al., 2006).

Desideratum 2 <Methodic Realization of Agents and Bidding Strategies>

The BidGenerator must offer methods for realizing agents and bidding strategies.

In order to develop bidding strategies one needs to solve the economic design (Section 3) and technical design issues. The technical design of the bidding agent framework requires that it provide well-defined implementation methods for both, the bidding agents and bidding strategies. Based on such a method, consumers and providers can setup and start their own agents and bidding strategies, reuse existing strategies or implement new ones. The methods for developing bidding agents and bidding strategies have to be compatible with existing agent framework standards like FIPA and MASIF, i.e., share commonly accepted principles and interfaces (Milojicic et al., 1998; FIPA, 2002b).

Desideratum 3 <Using Well-Defined Communication Protocols>

The BidGenerator must integrate and use a well-defined communication protocol to exchange private, public, contract and market information.

In order to communicate with the environment, bidding agents must implement and utilize well-defined communication protocols. The exchanged messages may have different formats, which depend on the message direction and communication context, i.e., between certain components. Moreover, communication protocols set the rules

for the interaction between bidding agents and the market – e.g., a bidding agent cannot receive a match without submitting a bid, nor can they submit a bid without receiving a bid request, etc. The content of a communication protocol (messages) has to be defined as commonly agreed upon concepts and properties, i.e. in an ontology, in order to improve the overall acceptance and its integration in the system (Fornara et al., 2007). In a market-based scheduling context, communication protocols have to include technical attributes like CPUs, memory, storage and bandwidth, as well as economic attributes like bid and duration (Windsor et al., 2009; Chevaleyre et al., 2006).

Desideratum 4 <Applicable in a Market-based Scheduling Domain>

The BidGenerator must support the automation of purchasing and provisioning processes of external computing services from the corresponding markets. Furthermore, BidGenerator must support simultaneous bid requests for computing services from consumers or providers (Windsor et al., 2009).

Desideratum 4 might also be valid in other domains. However, in a market-based scheduling context, the API of the *BidGenerator* has to offer interfaces for secured communication, the signing of bids and the processing of service contracts. In general, the adopted security, signing and contract mechanisms are more specific for the market-based scheduling domain than for other domains like mobile services, robotics and e-commerce.

Desideratum 5 <Integrating Market Information Service>

The BidGenerator must support the integration of a market information service, which provides information about other agents' bids and clearing prices.

Information about available auction mechanism and traded computing services has to be stored in registries (“green pages”) and made findable through a query language according to the technical specification required. The BidGenerator has to support integration with a market information service to query aggregated information of current, past bids and clearing prices for the required computing services (Brunner et al., 2008; Borissov et al., 2009a).

Desideratum 6 <Supporting Secured and Trusted Communication>

The BidGenerator must integrate appropriate security and trust mechanisms when communicating with the market.

Privacy, data protection and security has to be ensured with well-defined policies, which are enforced by an institution since applications like TXTDemand handle private and sensitive customer sales data (Windsor et al., 2009). The transfer of consumer data, as well as bidding processes need to be performed over a secured communication line. Security in the SORMA system has to be provided through a certified authentication process, which authorizes consumers and providers to interact with the SORMA market. Moreover, to ensure the legality of all submitted messages to the market, consumer and provider bids must be signed and validated before they can be considered as binding and being matched on the market (SORMA, 2008; Nimis et al., 2008, 2009).

These six requirements are the basis for developing and realizing the bidding agent framework (Section 4.4).

4.3 Existing Agent Frameworks

4.3.1 Selection of Existing Agent Frameworks

There are many agent frameworks in the existing literature and in practice. Some of them are more general, while others are designed to conduct specific tasks in a specific domain (mobile agents, agents for robots, e-commerce agents), but only few of them define interfaces explicitly for implementing bidding strategies and bidding agents that are applicable in a market-based scheduling domain. The relative straightforward number of existing surveys for agent frameworks often compare only specific criteria, provide a comparison for a certain selection of frameworks or are outdated (Nwana, 1996; Vrba, 2003; Bordini et al., 2006; Such et al., 2009). Vrba (2003) compares selected agent frameworks for their FIPA compatibility in terms of software interfaces and communication protocols, the application of security protocols, the type of license provided and the programming language of their implementation. Bordini et al. (2006) provides a brief description of selected agent frameworks without a direct comparison based on well-defined criteria. Such et al. (2009) focuses on the security capabilities of agent frameworks. The next two sections provide a detailed comparison of promising existing agent frameworks with regard to their applicability in a market-based scheduling domain according to the specified desiderata.

Table 4.1 presents an overview of existing agent frameworks pertaining to a standardized design method, a communication protocol and license type. Standardization efforts like *FIPA* and *MASIF* give best practices and policies for developing multi-agent systems and their interactions (FIPA, 2002b,a; Milojevic et al., 1998).

Table 4.1: Standardization of agent frameworks. ○ indicates that no explicit information was found ● indicates partial support of an agent framework property

Framework	Standard Compatible	Communication Protocol Support	License
FIPA-OS	FIPA	FIPA ACL	Open Source
JADE	FIPA	FIPA ACL	Open Source
Grasshopper	FIPA, MASIF	FIPA ACL, MASIF, Custom	Free for Non-commercial Use
Cougaar	●	Custom	Open Source
SHUFFLE	FIPA	FIPA ACL	○
LEAP	FIPA	FIPA ACL, Custom Protocols	Open Source
CRUMPET	FIPA	FIPA ACL	○
ADK	FIPA	FIPA ACL	Commercial
Aglets	MASIF	Custom	Open Source
JACK	●	Custom	Commercial
MAGMA	●	Custom Negotiation, Auction Protocols	○
AuctionBot	●	Custom Auction Protocols	○
e-Game	●	FIPA ACL	○
JCAT (JASA)	●	Custom Auction Protocols	Open Source

The *FIPA* community provides a general framework for developing agents and agent communication; *MASIF* has a similar focus, but for the domain of mobile agents. However, agent frameworks have to be in keeping with these best practices and extend them with the required additional functionalities.

The analysis excludes agent frameworks for which there is no available documentation, source code or anymore support, such as the *Java Agent Services*¹ initiative and

¹<<http://jcp.org/en/jsr/detail?id=087>>.

many older agent platforms like *Zeus*² and the *Comtec Agent Platform*.³

As shown in Table 4.1, many agent frameworks are compliant with the *FIPA* agent architecture specification, some comply with *MASIF* and others do not explicitly comply with any open standard. *FIPA-OS*⁴ and *JADE*⁵ are reference implementations of *FIPA*. For agents running on mobile devices, the *LEAP* (Bergenti and Poggi, 2002), *SHUFFLE* (Robles et al., 2001) and *CRUMPET* (Poslad et al., 2001) projects offer agent frameworks based on a lightweight version of the *FIPA* specification and its communication protocol. In addition, *FIPA* compatible frameworks often adopt *FIPA*'s agent communication language (ACL), which defines a protocol for implementing general agent communication procedures (FIPA, 2002a). Nearly half of the frameworks analyzed define and implement their own (domain-specific) communication protocols and provide their source code to the community.

This literature review concentrates on *Open Source* agent frameworks that offer public documentation of their software architecture, components and interface description in order to evaluate them for an application in a market-based scheduling domain.

4.3.2 Analytical Comparison to the Desiderata

Table 4.2 maps the specified desiderata to the agent frameworks analyzed. The agent frameworks are evaluated analytically according to their specifications and the specified desiderata for this work.

Decoupling of Market Platform and Bidding Agents

D1 requires that the market platform and agent frameworks be independent entities; the first is executed by an institution, and the second by the providers and consumers. The agent frameworks analyzed can be clustered into generic (e.g., *FIPA*-based) and domain-specific frameworks (e.g., mobile services, production robots, e-commerce). Agent frameworks like *FIPA-OS*, *JADE*, *Grasshopper*, *Cougar* and *Aglets* offer interfaces for implementing generic agents and interaction protocols that can be applied and reused in different fields (Helsing et al., 2004). These frameworks do not offer any methods or interfaces for implementing bidding strategies. *JACK* offers a generic agent framework for implementing lightweight and reusable agents, however, *JACK* is a proprietary framework, which is not free for use and adaptation (Howden et al.,

²<http://labs.bt.com/projects/agents/zeus>.

³<http://www.fipa.org/resources/livesystems.html>.

⁴<http://sourceforge.net/projects/fipa-os/>.

⁵<http://jade.tilab.com>.

Table 4.2: Comparison of the agent frameworks. D1 – Decoupling of Market Platform and Bidding Agents. D2 – Methodic Realization of Agents and Bidding Strategies. D3 – Communication Protocols. D4 – Market-based Scheduling Domain. D5 – Integrating Market Information Service. D6 – Secured and Trusted Communication. ✓ indicates explicit support ○ indicates that no explicit information was found or desideratum is not met. ● indicates partial support.

Framework	D1	D2	D3	D4	D5	D6
FIPA-OS	●	●	✓	●	●	●
JADE	●	●	✓	●	●	●
Grasshopper	●	●	✓	●	●	●
Cougaar	●	●	○	●	●	●
SHUFFLE	●	●	✓	●	●	●
LEAP	●	●	✓	●	●	●
CRUMPET	●	●	✓	●	●	●
ADK	●	✓	✓	●	✓	●
Aglets	●	●	✓	●	○	●
JACK	●	●	○	●	○	●
MAGMA	✓	●	○	●	✓	●
AuctionBot	✓	●	●	●	✓	●
e-Game	✓	●	✓	●	✓	●
JCAT (JASA)	✓	✓	●	●	✓	●
This work	✓	✓	✓	✓	✓	✓

2001). *AuctionBot* and *e-Game* focus on the realization of market platforms and the implementation of flexible market mechanisms, and offer generic interfaces to connect external agents (Wurman et al., 1998; Fasli and Michalakopoulos, 2008). The *Agent Development Kit*, *ADK*, focuses on agent design, agent logic and specification of the communication middleware, but not on interactions with market mechanisms (Xu and Shatz, 2003).

Only few frameworks, *MAGMA* and *JCAT*⁶, offer interfaces and methods to implement both market mechanisms and agents. *MAGMA* describes an overall infrastructure for building agent-based virtual marketplaces (Tsvetovatyy et al., 1997). It

⁶Used in the *Trading Agent Competition* (TAC), <<http://jcat.sourceforge.net>>.

clearly separates the market from the agents and describes a framework for implementing agents, however, the description remains on an abstract level and further implementation and application details are missing. The main design goal of *JCAT* is to provide a platform for the development and evaluation of market mechanisms and agent strategies in different tournament scenarios,⁷ e.g., TAC Travel, CAT, TAC SCM and TAC Ad Auctions (Cai et al., 2009). The *JCAT* agent framework provides interfaces and methods to develop, test and evaluate market mechanisms and bidding strategies.

Methodic Realization of Agents and Bidding Strategies

Most of the frameworks define interfaces for realizing agent logic, but few focus on the design and realization of bidding strategies. *FIPA-OS*, *JADE*, *Grasshopper*, *Cougaar* and *Aglets* offer interfaces to model and implement generic agents and their interactions, but there are no specifications for the realization of bidding strategies. *MAGMA* describes a process flow of bidding agents, but does not provide a specification of implemented and evaluated bidding strategies and how they coexist with the agents (Tsvetovatyy et al., 1997). The implementation of *ADK* agents is modular and defines the *Decision Making*, *Message Passing* and *Functional* units. The agent's logic goals, action plan and knowledge base are part of the *Decision Making* unit; the communication interfaces and interaction rules are part of the *Message Passing*, and the related low-level functionality is part of the *Functional* unit. *ADK* describes the general concepts of a *Decision Making* model and applies it in a *case study for air ticket trading*, however, Xu and Shatz (2003) presented a proof-of-concept realization of case study with the associated *ADK* agent interfaces, but an evaluation of bidding strategies and decision rules was not a part of their work. *e-Game* and *AuctionBot* focus on the design of market mechanisms and each of them provides platforms to realize them, but not concrete interfaces or implementations of bidding strategies. *JCAT* seems to be the only framework that provides well-defined interfaces to realize both agents and bidding strategies in a modular way. Furthermore, *JCAT* implements state-of-the-art auctions and bidding strategies.

Using Well-Defined Communication Protocols

The most of the analyzed frameworks apply communication protocols, which are based on *FIPA ACL (D3)*. However, some frameworks define their own custom communication protocols. The *FIPA ACL* is the candidate for describing general agent interactions (FIPA, 2002a). An *ACL message* specifies a well-defined structure

⁷<http://tradingagents.org>.

of concepts. *ACL's performatives* specify the intentions of the agents like *Propose*, *Accept Proposal*, *Reject Proposal*, *Confirm*, *Inform*, *Query* and others. Further concepts model the type of communication between agents like *sender*, *receiver* and *conversation-id*, the concept *content* enables the transfer of an encoded message in a user-defined format. There is a concept called *ontology*, which is associated with the *content* concept of the *ACL message* and expresses the meaning of the *content's* data. Finally, the *protocol* concept defines the rules (semantics) of the interactions with a specific set and order of *ACL messages*. However, the specifications for the *ACL* concepts, *content* and *protocol*, are not finalized and marked as deprecated.⁸ *ACL* defines general *performatives* that support negotiation protocols, but there are no concrete specifications for auctions, bidding concepts, service level agreements, payment methods, penalties or support for market-based scheduling. Furthermore, an *ACL message* defines many concepts, but with missing details and use cases for their applications in real systems. For example, transferring an *ontology* in addition to the *content* part decreases the communication tractability of the system since each of the messages has to be interpreted and validated with an *ontology reasoner* before using the actual data of the *content*.

As one of the novel frameworks, only *e-Game* supports *ACL*. Agents connect to the *e-Game* platform through TCP and exchange messages using a subset of the *ACL* language and its performatives (Fasli and Michalakopoulos, 2008). *JCAT* is platform for evaluating bidding strategies and market mechanisms in tournament scenarios. Communication of agents' intentions is orchestrated with the *CATP* communication protocol (Niu et al., 2009). *CATP* consists of concepts to connect, send bids and receive market information, but its overall focus is to model the interaction and characteristics of tournaments with "performatives" like *gamestarting*, *gamestarted*, *gameover*, *dayopening*, *dayclosed*, *roundopened* and *roundclosed*. Furthermore, it does not provide concrete specification for the transaction objects, therefore, the market performs only price-based matchmaking, but not matchmaking of technical descriptions as required for computing services.

Applicable in a Market-Based Scheduling Domain

The presented agent platforms are well known and accepted in the agent research. These frameworks are designed and implemented for a specific domain like mobile services, e-commerce and tournaments, however, there is no agent framework available that is designed and evaluated for a market-based scheduling domain. Theoretically,

⁸See <http://www.fipa.org/specs/fipa00007>. and <http://www.fipa.org/specs/fipa00025>.

each agent framework can be extended with the required additional modules, however, the different frameworks are designed with their own philosophy and goals for their specific domain, which also affect the design of their interfaces and interaction protocols, e.g., for tournaments, mobile services or e-commerce. Thus, practically, an “extension” with the required additional modules is usually required to change the modules that are already available as well. Moreover, most of these frameworks are mainly developed and evaluated as proof of concepts, but not realized in real systems.

Integrating Market Information Service

In a market-based scheduling scenario, agents will need to query information about available services and their technical specifications. Furthermore, agents will need to request market information on available actions for these services, as well as current and past bids in order to derive their bidding strategy for the automated purchasing and provisioning of the required computing services.

The *FIPA ACL* specifies concepts for exchanging user-defined information between agents. Therefore, agent frameworks which adopt an ACL are, in principle, able to exchange market information, however, only few of the frameworks analyzed define the exchange of market information explicitly. Therefore, the frameworks that implement the ACL, but do not explicitly define market information, have been marked as partial support.

ADK integrates the discovery, join, and lookup mechanisms of the *Jini*⁹ system, so agents can find, register for and invoke services and service information. The registered agents receive market information (e.g., for negotiating airline tickets) from other agents and can react to them according to the negotiation protocols implemented. In *MAGMA*, the role of market information management is provided by the *Advertising Server*. Agents in *MAGMA* send ads for what they offer or what they are looking for. Thus, agents can query the *Advertising Server* about available ads, their descriptions and prices. *MAGMA* discusses the concept of automatic negotiation by using such market information to decide on the selection of a service based on the ads that match the agent’s criteria.

As platforms for developing market mechanisms, the *AuctionBot*, *e-Game* and *JCAT* frameworks explicitly define policies for exchanging market information between the participating agents in the auctions.

⁹<http://www.jini.org>.

Supporting Secured and Trusted Communication

According to *D6*, consumer data and bids need to be transferred over a secured communication line, where the security mechanism is a part of the SORMA system. Moreover, consumer and provider bids must be signed and validated in order to be considered as binding when they arrive in the order book (Nimis et al., 2008, 2009). All of the agent frameworks presented integrate security facilities implicitly when using standard symmetric or asymmetric internet protocols of the ISO security layer like IPsec, SSL, TLS and Kerberos (Such et al., 2009). These security protocols offer features for the authentication, integrity and encryption of data. *FIPA* started working on agent security¹⁰ at the end of the 1990s, but a final specification has not been published yet. Navarro and Borrell (2006) and Demchenko et al. (2007) propose the application of the OASIS standards *eXtensible Access Control Markup Language* (XACML) and *Secure Assertion Markup Language* (SAML) in distributed systems like Grids. Moreover, SAML is adopted and well applied in the Cloud computing domain (Jansen and Grance, 2011; Jensen et al., 2009). XACML focuses on the specification of control policies for authorizing users and the enforcement of their access rights (read/write) for objects. SAML provides authentication and authorization mechanisms for users and secured data exchange. SAML is well applied in distributed systems. One or several identity providers (IDP) are responsible for the management of users and their access rights. In order to interact with the system, a user has to authenticate and request an (timely limited) access token (called assertion) from an IDP and a service provider. Each of the user messages and data is signed with the user's personal signature, which is part of the user access token. SAML is assigned to the so-called single sign-on (SSO) mechanism (Armando et al., 2008).

SAML and XACML are well adopted in distributed systems and in securing the exchange of data, however, security and trust mechanisms are still unexplored in conjunction with market mechanisms and bidding agents for a market-based scheduling domain, especially for the *automation of bidding processes*. Moreover, the agent frameworks investigated in Table 4.2 do not implement either XACML or SAML.

The *ADK*, *e-Game* and *JCAT* agent frameworks are good candidates that satisfy at least three of the six desiderata specified. However, *e-Game* focuses entirely on the design of market mechanisms and its source code implementation is not open. None of these or the other agent frameworks are specifically modeled or evaluated for a market-based scheduling domain and none of them implement common security mechanisms for distributed systems like SAML and XACML. Moreover, *ADK*

¹⁰<http://www.fipa.org/specs/fipa00020/0C00020A.html>.

focuses on agent design rather than market mechanisms and *JCAT* is designed to be executed in tournament scenarios. Theoretically, each agent framework can be extended with the missing modules required, however, the different frameworks are motivated and designed according to the specific goals of their target domains, which affect the definition and implementation of their interfaces and interaction paradigm. In practice, extending a framework with additional modules usually requires changing the existing ones. This work requires an agent framework that is applicable in a real system for market-based scheduling and that fully satisfies the specified desiderata according to the application scenarios in Section 2.4.1 and the SORMA project (Nimis et al., 2008, 2009). The next section presents the *BidGenerator*, a novel agent framework for automated bidding as one of the contributions of this work.

4.4 Architecture of the Bidding Agent Framework

This section presents the logical and dynamic design architecture of the *BidGenerator* framework. The design and implementation details of *BidGenerator* are modeled with UML, which is common in the agent community and offers various standardized diagrams for displaying the components, their interdependencies, internal structure and interactions (Bresciani et al., 2004; Zambonelli et al., 2003; Bauer and Odell, 2005; FIPA, 2007). The presentation of the different aspects of the *BidGenerator* framework is organized according to the specified design desiderata by enhancing the overall bidding scenario in Section 2.4.4.

4.4.1 Decoupling of Market Platform and Bidding Agents

As discussed in Section 2.4.4, the general role of the *BidGenerator* framework is to be the binding element between the consumer's *Application Orchestrator* and the market platform or the provider's *Resource Orchestrator* and the market platform. In its role, the *BidGenerator* provides well-defined interfaces to implement and run bidding agents. Figure 4.1 presents an integrated view of the *BidGenerator* and its related components. This view clearly differentiates between components on the client side that installed locally on the consumer and provider sides, and the market platform component *Communication Space (CSpace)*. The latter is executed by a third-party institution together with the other related components, which are part of the SORMA market middleware (Nimis et al., 2009). The interfaces, which realize the communication between *BidGenerator* and *CSpace* are implemented within the market connector component called *CSpaceConnector*.

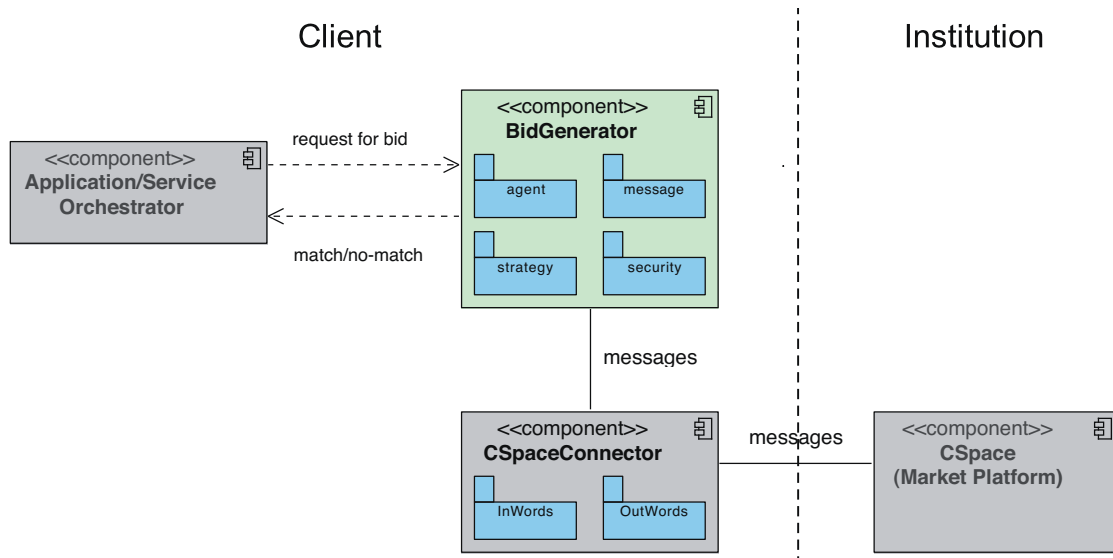


Figure 4.1: A bird’s-eye view of the interactions between BidGenerator and related components

The clients – *Application Orchestrator* and the *Resource Orchestrator* – communicate directly with *BidGenerator* by submitting “request for bid” (*bidCall*) messages with the required or provided computing service descriptions and economic information like their valuation, payment method and preferred bidding strategy implementation that are all part of the *preferences* message (for more details see Section 5.3.1). The “match/no-match” messages (from the *MarketMessage* type) are asynchronously returned back from the market to the *Application Orchestrator* and the *Resource Orchestrator* through the *BidGenerator*’s invocation response web interface:

```
MarketMessage bidCall(credential:Credential, prefs:PrivateMessage)
```

Each client’s Web service invocation of *bidCall* is conducted with a username and password, which is part of the *credential* object and validated from the security component on each interaction with the SORMA system. The *BidGenerator* provides interfaces and implementations for *agents* and bidding *strategies*, and generates and exchanges *messages* with the market through *secured* communication line. The *BidGenerator* submits messages to the target market mechanism (e.g., CDA), as part of the CSpace platform, by implementing the CDA’s *InWords*¹¹ interface and receives

¹¹messages send from the agents to the market mechanism

messages with the CDA's *OutWords*¹² interface. *CSpace* is the market platform of the SORMA project, which provides a method and interfaces to implement market protocols like *English*, *Vickrey* and *CDA* auctions (Nimis et al., 2009). *CSpace* enables multiple bidding agents to connect and exchange messages according to the defined rules of the implemented market mechanism. The concept *Conversation* of *CSpace* is used to emphasize the general specification of the platform for implementing different market protocols, and also any other protocol where agents are involved and need to communicate. Security and trust in *CSpace* are parts of its design philosophy and architecture. Secured communication is guaranteed by the SORMA system, so *CSpace* runs in a secured environment. Trust is one of the main characteristics of the *CSpace* platform. Each of the implemented market protocols is an autonomous unit, which can be uploaded and downloaded in the *CSpace* platform. Each version of a market mechanism has a checksum, which can be verified by any of the participants and controlling institutions. Each of the "conversations" between the market participants and market mechanisms are recorded and can be verified by a public notary institution at any time or replayed in order to prove and validate the outcome of the matchmaking process. The outcome of the matchmaking results in the creation of signed and binding contracts between consumers and providers. The design philosophy of the *CSpace* platform assumes that users will use market protocols that they trust. *CSpace* does not explicitly control the types and implementation of the market protocols, but it is assumed that in long-term "unfair, malicious and erroneous protocols are rejected by the users in a similar way as such contracts are rejected in real life" (Nimis et al., 2009).

4.4.2 Methodic Realization of Agents and Bidding Strategies

The overall architecture of the *BidGenerator* framework is depicted as a class diagram in Figure 4.2. It specifies and implements interfaces for five main packages – *agent*, *strategy*, *learner*, *message* and *security*.

The *agent* package provides core interfaces for implementing (bidding) agents, as well as reference implementations of consumer and provider agents, which have been developed, integrated, tested and evaluated as part of the SORMA system. *BidGenerator's strategy* package provides well-defined interfaces for implementing bidding strategies and a method for a dynamic assignment of a bidding strategy implementation to an

¹²messages send from the market mechanism to the agents

agent according to the client's preferences. The *BidGenerator* implements state-of-the-art bidding strategies like ZIP and GD, as well as the Q-Strategy developed here (Sections 3.3.4 and 3.4). The clients report the class name of the preferred bidding strategy as part of their *PrivateMessages*. Each of the realized bidding strategies implements an interface method for receiving and transforming clients' *PrivateMessages* into bid messages (*PublicMessages*):

```
PublicMessage generateBid(PrivateMessage request)
```

The well-defined interfaces of the *strategy* package allow a straightforward development and application of novel user-defined bidding strategies. The *learner* package of the *BidGenerator* framework specifies an interface for realizing well-known learning algorithms. The separation of the packages' *strategy* and *learner* is motivated by the fact that a bidding strategy may implement several decision steps of information gathering (e.g., current prices, other agents' bids and matches), information aggregation and bid generation. Each of these steps may utilize different aggregation algorithms (clustering, trend analysis, decision tree, reinforcement learning, transfer learning and others) for estimating the decision parameters of the bid generation function (Stone, 2007a). The method *newState* creates or updates state information for the executed actions and the received reward, which is called score, and the *bestAction* method retrieves the current *best action* with the highest aggregated¹³ score from the strategy's knowledge base.

The *message* package of the *BidGenerator* framework offers interfaces and reference implementations for the exchanged messages in the system. The specifications of the message types *PrivateMessage*, *PublicMessage*, *MarketMessage*, *StateMessage* and *MarketInformation* are provided in Section 5.3. The *MessageImpl* class implements the management of the bidding-related messages for each of the consumer and provider requests. The reward from each consumer's or provider's action is measured with their defined scoring functions (*ConsumerScoringFunction* and *ProviderScoringFunction*) and it is used to update the knowledge base of the *BidGenerator* framework. The structure of the knowledge base is specific for each of the realized bidding strategies and depends on the definition of their information model. Currently, *BidGenerator* implements an in memory knowledge base, but the information models of the different bidding strategies can be mapped to persistent storage with relatively low effort.

¹³The aggregation function is based on a pre-defined rule of the learning algorithm.

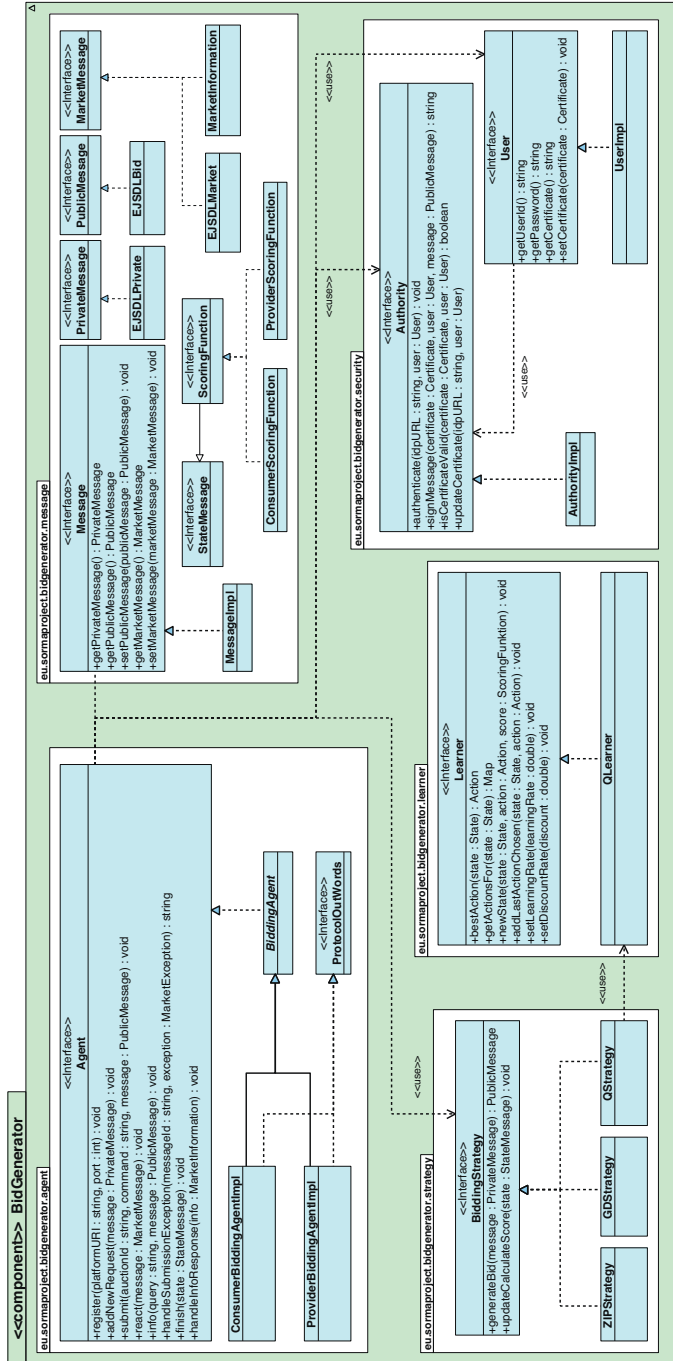


Figure 4.2: BidGenerator architecture

The last package of the *BidGenerator* framework is *security*. This package offers functionality for credentials management, authentication and the signature of bids, which are required to communicate with the trusted SORMA middleware. The security assertions are requested from a trusted identity provider, which generates proxy certificates based on the requestor's credentials.

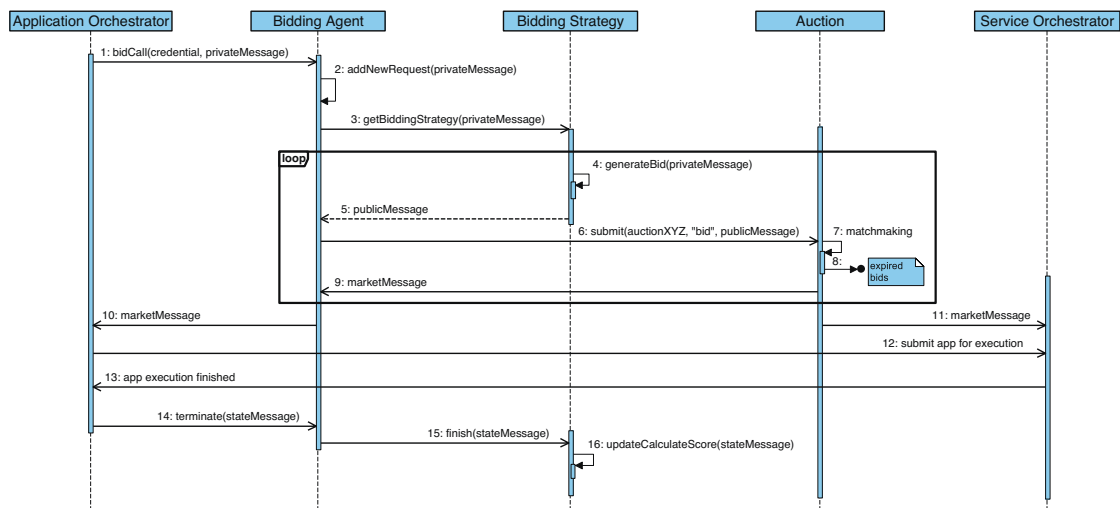


Figure 4.3: BidGenerator's dynamic view of bidding processes

Figure 4.3 presents a sequence diagram, a dynamic view of bid generation processes from the consumer's perspective. The provider case is analogous to the consumer case. The *Application Orchestrator* has knowledge about the consumer's applications, which require provider machines in order to be executed. The consumer's credentials in terms of user name and password, as well as the application's technical and economic preferences (*privateMessage*) are submitted in to the *Bidding Agent* in step 1. The *Bidding Agent* registers the new request in accordance with the preferences, initializes the preferred *Bidding Strategy* (steps 2 and 3) and starts the bidding process (steps 4 to 9). The bid, *publicMessage*, is generated in step 4 and the result is sent back to the *Bidding Agent* in step 5. In step 6 the bid is signed with the consumer's signature, which is provided by the trusted identity provider and sent to the target *Auction*. The *Auction* performs technical and economic matchmaking and response with the successful match, *marketMessage*, back to the winning *Bidding Agents* of the consumer and provider (steps 9, 10 and 11). The *marketMessage* is signed by both the consumer and provider. The signatures are automatically verified from the security component with the trusted identity provider. In the event that

the consumer or provider bids do not match, they remain stored in the *Auction*'s order book and are removed when their limit times expire (step 8). The *Application Orchestrator* receives the allocation (*marketMessage*) and submits the consumer application to the allocated machine assigned by the provider's *Service Orchestrator* (step 12). The *Application Orchestrator* monitors the execution and completion of its application and at the end, it passes the final state of the application execution (values of the monitored key performance indicators) to the *Bidding Agent* (steps 13 and 14). The *Bidding Agent* uses the state information to calculate the reward (score) of the application execution (step 15). The score is calculated according to the consumer's scoring function. The score value updates the consumer's knowledge base according to the selected consumer preferences (step 16), e.g., application's valuation, selected bidding strategy, scoring function, etc.

4.4.3 Using Well-Defined Communication Protocols

The *BidGenerator* integrates a well-defined communication protocol for exchanging preferences, bids and market messages of technical and economic attributes (Borissov et al., 2009b). A detailed description of the communication protocol is presented in Section 5.3.

The interaction with the *Auction* in *CSpace* is governed according to well-defined rules, which define the type and content of the incoming messages to the market and the outgoing messages from the market. These rules are specified within the *ProtocolInWords* and *ProtocolOutWords* interfaces, which are part of the *CSpaceConnector* component (Figure 4.4). These two interfaces are provided for each of the market mechanisms implemented in *CSpace*. *ProtocolInWords* is implemented within the market mechanism and used by the *Bidding Agent* to send messages like "bid" and "info" to the target auction, e.g., CDA. The "bid" command places a binding bid in the order book, and the "info" command queries market information about the last N bids, offers and prices for the specified transaction object description, which are part of the *PublicMessage*. The types of messages submitted from the market to the *Bidding Agent* are specified within the *ProtocolOutWords* interface. Therefore, the *Bidding Agents* implement the *ProtocolOutWords* interface in order to receive and react to the market messages.

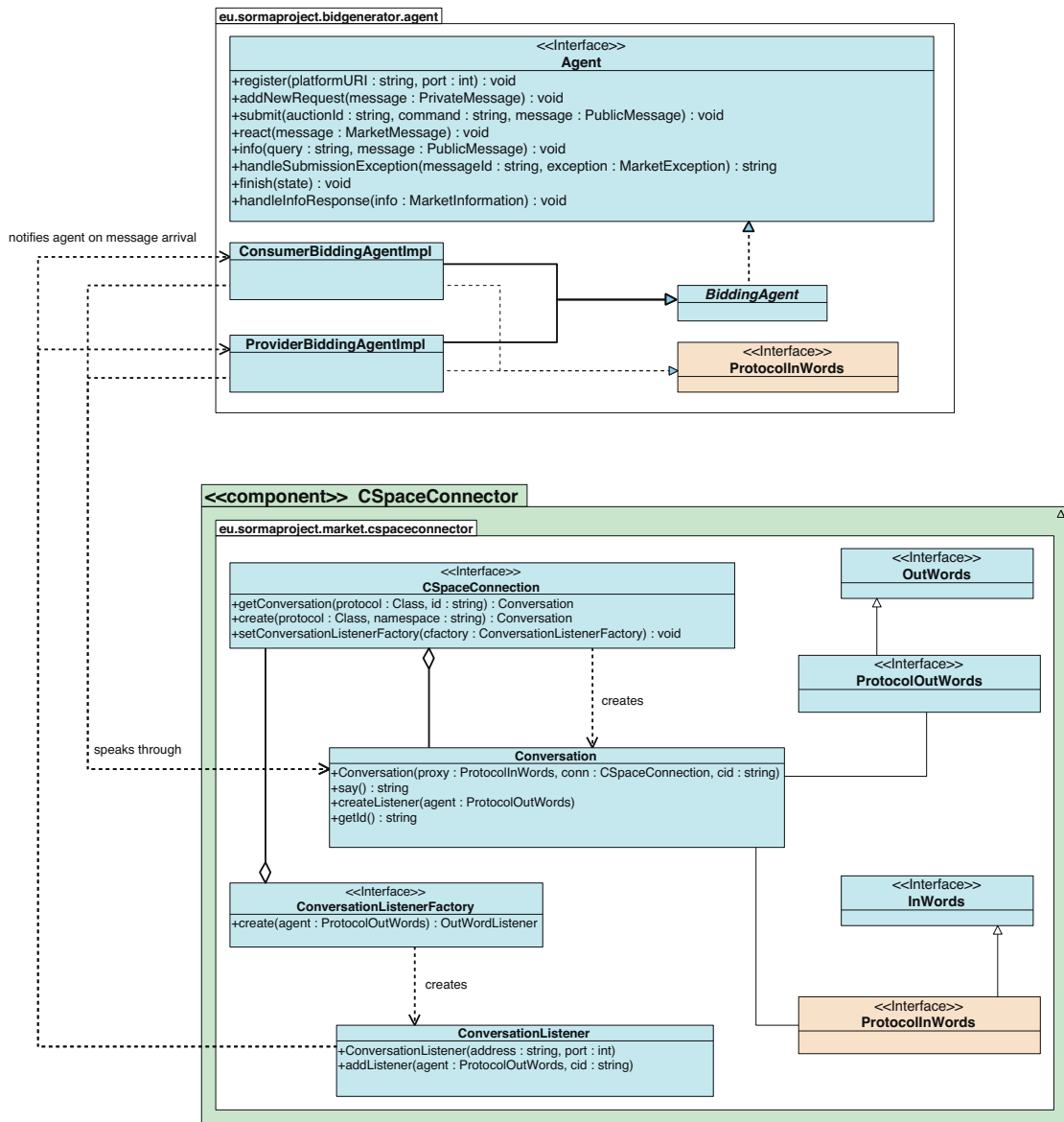


Figure 4.4: BidGenerator communication components

4.4.4 Applicable in a Market-based Scheduling Domain

BidGenerator is a general framework for implementing agents, bidding strategies and their interactions with market mechanisms. However, it integrates a communication protocol and a security mechanism, which are specific to a market-based scheduling domain. The message types of SORMA’s communication protocol contain technical

attributes for describing computing service configurations, e.g., for CPU, memory and storage. Borissov et al. (2009b) demonstrated a mechanism for creating service level agreements of technical and economic attributes based on a specified communication protocol. Moreover, the *BidGenerator* was integrated, tested and evaluated in the SORMA system according to the application scenarios presented in Section 2.4.1.

SAML is the integrated security protocol in SORMA. As a single-sign-on service, SAML is applicable in any distributed system, which requires trusted authentication and authorization.¹⁴ *BidGenerator* is especially designed for a market-based scheduling domain by realizing interfaces for secured single sign-on authentication and the exchange of signed bids and matches in that domain.

4.4.5 Integrating Market Information Service

Bergemann and Pesendorfer (2007) discuss the importance of market information to deduce initial prices and optimal trading times for the demanded or offered transaction objects in a market. Sophisticated bidding strategies can be designed to perform common trend extrapolation methods for prices and submission times in order to optimize bidding processes (Borissov et al., 2009a). Well-known bidding strategies like ZIP and GD generate their bids from algorithms, which aggregate market information of current and past bids and prices (Vytelingum et al., 2008; Gjerstad, 2003; Das et al., 2001). Therefore, the information provided by market mechanisms is important for the design of bidding agents and strategies. In the context of SORMA, there are two types of market information – *MarketMessage* and *MarketInformation*. *MarketMessage* is the market message that is created when there is a match between a consumer and provider bids. *MarketMessage* is private and available only to the respective consumers and providers; other agents do not have access to it. The *MarketInformation* message contains the id of the auction and lists of the last N consumers' and providers' bids and prices for the target transaction object.

In order to submit a bid, the agent has to know the id of the auction mechanism that trades the desired transaction object type (e.g., computing service with 1 CPU, 10GB memory and 64GB storage). The auction id is requested with the *info* method, which queries the market platform (*CSpace*) for available auctions that match the technical description in the attached *PublicMessage*. The response from this query

¹⁴SAML is also adopted in the Google's AppEngine: <http://code.google.com/googleapps/domain/sso/saml_reference_implementation.html>.

is returned with the *handleInfoResponse* method as part of the *MarketInformation* message.

```
void info(string query, PublicMessage message) : void
void handleInfoResponse(MarketInformation info)
```

Furthermore, market information of other agents' bids and prices is also provided with the *info* method for the specified technical description in *PublicMessage* and a query, which requests the last N bids and prices for the target transaction object (Section 5.3.5). The *MarketInformation* returns empty lists in case there is not a running auction for the requested transaction object type. The *info* and *handleInfoResponse* methods are non-blocking, thus the bidding agent can perform further consumer or provider requests until the response from the market is received.

```
void submit(string auctionId, string command, PublicMessage message)
void react(MarketMessage message)
```

Bidding agents in *BidGenerator* implement the submission of bids (*PublicMessage*) and the reaction of matches (*MarketMessage*) with the *submit* and *react* methods of the agent interface. Similar to the *info* method, these methods are executed asynchronously so that the bidding agent can proceed with the bid generation processes of the other requests.

4.4.6 Supporting Secured and Trusted Communication

SORMA's security component provides a single sign-on mechanism for authenticating consumers' and providers' bidding agents to communicate with the trusted SORMA market middleware. The security component is designed to provide distributed identity management in order to prevent single points of failures and to be applied in different geographic regions and their specific (legal) policies. The distributed single sign-on identity management in SORMA was implemented with the SAML protocol (SORMA, 2008; Nimis et al., 2009). The identities of the consumers and providers are stored and managed by (distributed) *identity providers* (idP). Before a consumer or provider can communicate with the SORMA market middleware it has to authenticate with the idP service. The idP issues the requester a security token called SAMLAssertion, which is unique and limited in time, and allows the requester to identify and communicate with the SORMA market middleware. Moreover, the idP

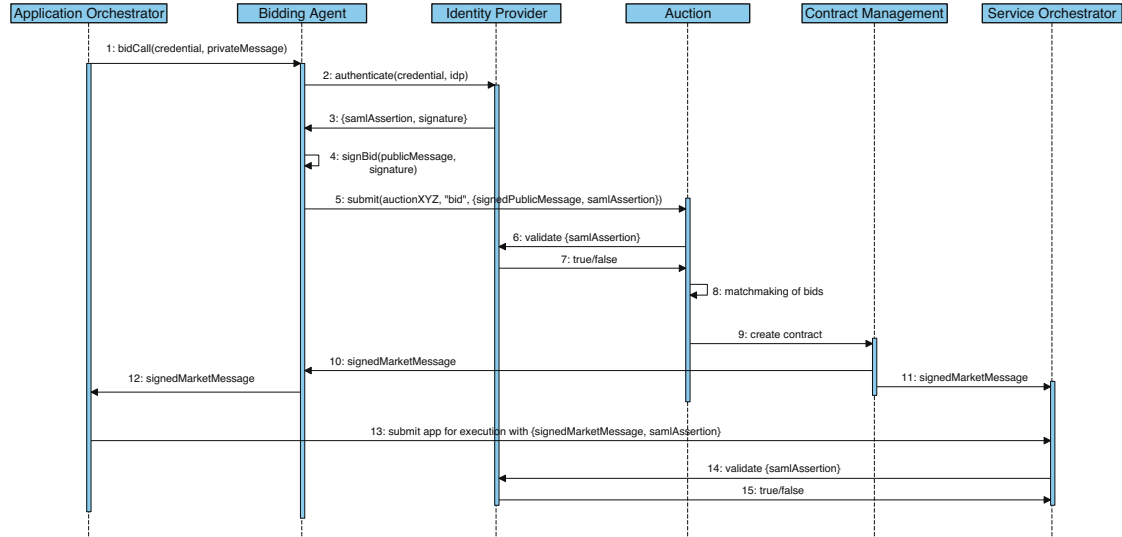


Figure 4.5: BidGenerator security integration

creates a unique signature for the requester, which is used to sign the submitted messages to *binding* and *legal* bids, as well as contracts.

The overall security process in the SORMA system is depicted as a sequence diagram in Figure 4.5 and complementary to Figure 4.3, but which focuses on the implemented security and trust mechanism. In step 1, the *Application Orchestrator* invokes the *Bidding Agent* with the submission of its *PrivateMessage* and credentials (user name and password). In step 2, the *Bidding Agent* uses the credentials to authenticate the requester with the idP and receives the required security token and signature to communicate with the SORMA market middleware. Based on the received *samlAssertion* and *signature* for the requester, the *Bidding Agent* signs the generated bid. In step 5, the signed *PublicMessage* and *samlAssertion* are submitted to the *Auction*. The *Auction* validates the bid submission with the bidder's *samlAssertion* by invoking the *validate* method of the idP (step 6). The provider returns true if the *samlAssertion* is valid and false, if not (step 7). If false, the bidder's *PrivateMessage* is canceled by the *Auction* and not considered in the matchmaking process. The *Auction* performs an economic and technical matchmaking of signed consumer and provider bids (step 8). When there is a match, the *Contract Management* creates a contract of the *MarketMessage* provided by the *Auction* and signs the *MarketMessage* with the target consumer and provider signatures (step 9). The signed *MarketMessage* is returned back to the *Application Orchestrator* and *Service Orchestrator* through their *Bidding Agents* (steps 10, 11 and 12). The *Application Orchestrator* submits the target appli-

cation to the provider's *Service Orchestrator* together with the signed *MarketMessage* and the consumer's *samlAssertion* (step 13). The *Service Orchestrator* validates the consumer's *samlAssertion* with the idP and its own copy of the signed *MarketMessage* (step 14). The *Service Orchestrator* deploys and executes the application when the idP response is true (step 15). When the idP validation response is false, the *Service Orchestrator* cancels the procedure of the consumer request.

4.5 Summary

This chapter presented the architecture of the *BidGenerator* framework, which enables a coexisting design and realization of agents and bidding strategies. The *BidGenerator* framework integrates facilities for information gathering and aggregation, as well as proactive and reactive communication with market mechanisms or other agents (cf. Definition 4.1.2). The decision making processes for trading transaction objects are executed with the selected bidding strategy, which is part of the *BidGenerator*'s strategy pool. The chapter starts with the specification of the design desiderata for developing bidding agents in a market-based scheduling domain. The analysis of existing agent frameworks shows that there is no available framework, which satisfies the desiderata.

The *BidGenerator* provides clear interfaces for implementing agents and bidding strategies. The interfaces of the *BidGenerator* framework are consistent with FIPA's reference specifications for agent design and communication (FIPA, 2002b). Adequate methods like *register*, *submit*, *info* and *react* ease the development of (bidding) agents with the *BidGenerator* framework. The *BidGenerator* is executed as a stand-alone component on the provider and consumer side, which is decoupled from the market platform (*D1*). Together with *D1*, *D2* allows consumers and providers to implement and configure their own agents and bidding strategies, and execute them in their private environments according to their preferences. Furthermore, the *BidGenerator* provides a realization of the specified interfaces and implements state-of-the-art bidding strategies and the *Q-Strategy*. The bidding agents of *BidGenerator* communicate with the market platform according to a well-defined communication protocol (*D3*). The communication interface is implemented as part of the *CSpaceConnector* component and the communication protocol is specified in Section 5.3. The *BidGenerator*'s architecture is specifically constructed for application in a market-based scheduling domain (*D4*). This consideration also affects the design, integration and realization of the related market information service (*D5*), as well as the security and trust mechanism (*D6*). Moreover, the *BidGenerator* framework was developed,

integrated, tested and evaluated as part of the SORMA system. A detailed technical analysis of the *BidGenerator*'s role in SORMA, as well as a performance analysis of the integrated components is presented in Chapter 7.

Chapter 5

Communication Protocols in Computing Service Markets

THE previous chapter presented the architecture of the BidGenerator framework. This chapter focuses on the design and realization of the communication protocol for agent interactions and messaging as part of Research Question 3 (*Communication Protocols*) in Section 1.2.

Communication protocols are core design issues in computing systems. In a market context, communication protocols define the interfaces for the message exchange, the rules for timing, bidding, information revelation and clearing, which are passed between the agents or related software components (Rosenschein and Zlotkin, 1994; Wurman, 2001). Examples of communication methods are *bid*, *update-bid*, *match* and *info*; example rules are the bidding constraints in terms of value ranges, the visibility of bids to other agents and the right of bid withdrawal. In a market-based scheduling context, a communication protocol (also called *bidding language* here) defines the content and format of the exchanged messages in terms of technical and economic attributes. Moreover, in a multi-agent environment, such data formats have to be syntactically and semantically well defined in order to reach a common understanding, wide acceptance and standardization.

Section 5.1 derives design desiderata for communication protocols in a market-based scheduling context. Section 5.2 presents existing communication protocols and compares them with the introduced desiderata. Section 5.3 presents a novel message protocol for market-based scheduling as one of the main contributions of this work. Section 5.4 formalizes the adopted matchmaking mechanism for the bids. Matchmaking mechanisms are not a part of the research of this work, however, they are part of the SORMA market (Nimis et al., 2009). Section 5.6 discusses extensions to support combinatorial bids for additional market-based scheduling scenarios and Section 5.7 summarizes the chapter.

5.1 Design Desiderata

The efficiency of the communication process depends on the number of exchanges required to reach an agreement between two parties (Endriss and Maudet, 2004). In a negotiation scenario, the exchange processes include communication calls like *propose*, *accept*, *reject* and *deal*. Therefore, the effort for reaching an agreement would require at least three communication calls between two parties (A sends a *propose* call to B, B sends an *accept* call to A, A sends a *deal* call to B). The number of required communication calls also depends on consumers' and providers' valuations, their bidding strategies and technical descriptions of the demanded and provided computing services. In a double auction mechanism, the number of communication calls can be reduced to two – *bid* and *match*. In a distributed scenario for market-based scheduling it is crucial to design a communication protocol which is *Communicationally Tractable*, i.e., efficient in the number of exchanged messages enabling efficient matchmaking capabilities.

The following design desiderata are derived from the application scenarios in Section 2.4.1, the technical challenges in Section 2.4.2, the bidding scenario in Section 2.4 and from the respective literature.

Desideratum 1 <Expressing Economic and Technical Preferences>

Bidding languages for market-based scheduling must allow for the expression of preferences, which consists of both technical and economic attributes.

The expressiveness and simplicity of a bidding language in a market-based scheduling context depends on the level of abstraction selected (e.g., ISO/OSI reference model; IaaS, PaaS or SaaS) for the design of the specific language. For example, the number of technical attributes can be significantly reduced by abstracting computing service characteristics like CPU architecture, CPU frequency, cache, and memory through performance benchmarks of *test applications* (Jain, 1991; Ostermann et al., 2010). Furthermore, the expressiveness and simplicity of a bidding language depends on whether the market mechanism supports atomic or combinatorial bids, where the latter introduces additional logical primitives like XOR, OR, AND and combinations thereof by way of preferences (Nisan, 2006). On the one hand, the expressiveness of a bidding language determines the complexity for the bidding parties to specify their preferences, on the other hand, it determines the complexity for matchmaking bids, which is an NP-Hard problem in the combinatorial case (Parsons and Klein, 2009). The specified application scenarios in Section 2.4 demand a bidding language for exchanging technical and economic attributes, which are combined in a common message format.

Desideratum 2 <Separation of Private, Public and Market Messages>

Bidding languages for market-based scheduling must support the specification of different message types for exchanging private, public, contract and market information.

Economic theory distinguishes between goods with a common value to all agents like a USB stick with 16 GB of storage space and those with private values, i.e., the value of a good differs from agent to agent (Lomuscio et al., 2003; Collins et al., 1998). Therefore, valuations for goods may be common or private knowledge. In a market-based scheduling scenario, different providers offer computing services on different hardware infrastructures that vary in the quality of service; and consumers also have different applications and configuration requirements for computing services.¹ In the case of private valuations, agents specify their own scoring functions and generate their bids according to their selected bidding strategy. The agent's outcome is measured according to the rewards received as a result of their actions and scoring functions. The bids of the consumers and providers are functions of their private valuations (Nisan, 2006). The bids are submitted to the market, therefore, they are marked as public. Agents' bid generation functions may also take current market information about other agents' bids and matches into account (Lomuscio et al., 2003). Furthermore, the agents may also exploit their experiences in past interactions with the bid generation function. Thus, markets for computing services have to offer message formats for exchanging market information on current and past bids, and clearing prices.

Desideratum 3 <Support the Creation of Service Contracts>

Bidding languages for market-based scheduling must support the creation of legally binding service contracts.

The result of a matchmaking process in a system for market-based scheduling is the creation of the service contract (service level agreement) between a consumer and a provider. A service level agreement has to include information about the duration of an application, the duration of the computing service provided, the payment method and information about compensation payments, also called penalties (Windsor et al., 2009). The payment method specifies the rules and type of the payments payment, such as prepayment, post-payment, pay per use and subscription (Windsor et al.,

¹Symantec (2008) reported that, on average, companies work with more than one thousand applications; almost all of the computing centers on the Top500 list (<www.top500.org>) are built with different hardware specifications.

2009; Weinhardt et al., 2009). The penalty part of a contract specifies the formalism for calculating penalty payments if a contract is not fulfilled (Kephart and Chess, 2003; Becker et al., 2008). Contracts are legal entities and require the signature of both the target consumer and target provider in order to be considered legally binding (see also Desideratum 6 in Section 4.2).

Desideratum 4 *◀Providing Semantic Descriptions of Market Concepts▶*

Bidding languages for market-based scheduling must provide a machine-readable semantic description for each of the adopted domain concepts and relations as part of their technical and economic attributes.

Desideratum 2 requires that information that is available only to owners (private), submitted to the market (public), and the market's response (a match or information about other agents' bids and matches) be clearly specified and separate. These message types include concepts with different semantics like valuation (private), bid (public) and clearing price (match). The different message types need to be well defined not only in structured models for the exchange of data like in XML, but also semantically in order to enable a common understanding of the machine-processable data in distributed complex systems for market-based scheduling.

An ontology provides a structured description of the protocol's concepts and relations that can be read by machines and humans. A formal description of the semantics of market concepts adopted in communication protocols is crucial to make sure that they understand each other, as well as to ensure their interoperability, verification, and integration in existing and new components. This is especially important for interdisciplinary research in the area of multi-agent systems and market mechanisms. The definition of domain ontologies in complex systems facilitates software engineering and integration processes in terms of a clear mapping between communication protocols and components' concepts in a complex distributed system, interoperability support through clearly structured and shared knowledge, reusability of already defined concepts, verification against existing concepts and their relations in other ontologies (Gasević et al., 2009; Tran and Low, 2008; Tamma et al., 2005). A best practice in ontology design is to map the ontology concepts of a market-based scheduling domain to concepts of well-known upper ontologies like *OpenCyc* (Mascardi et al., 2007). Modern semantic technologies like SAWSDL² enable the coexistence of XML-based data models and their explicit specification in ontologies (Lathem et al., 2007;

²Semantic Annotations for WSDL, <<http://www.w3.org/TR/sawSDL>>.

Kopeccky et al., 2007). On the one hand, this enables efficient data transfer with commonly applied XML technologies; on the other hand, it enables reasoning and efficient integration with other data models and their ontologies.

5.2 Existing Communication Protocols

Table 5.1 shows a selection of prominent communication protocols, which represent the agent, negotiation and bidding domains. The communication protocols are analytically compared according to the derived design desiderata.

Table 5.1: Comparison of communication protocols. D1 – Economic and Technical Preferences. D2 – Separation of Private, Public and Market Messages. D3 – Creation of Service Contracts. D4 – Semantic Descriptions of Market Concepts. ✓ indicates explicit support. ○ indicates that no explicit information was found or desideratum is not met. ● indicates partial support.

Communication Protocol	D1	D2	D3	D4
ACL	●	○	○	●
CATP	●	○	●	○
ClassAds	●	○	○	○
JSDL/JDL/GLUE/RSL	●	○	○	○
CIM	●	○	○	●
OVF/SDD	●	○	○	○
WS-Agreement	✓	○	✓	○
Nisan (2006)	●	○	○	○
MACE	✓	○	●	○
Vilajosana et al. (2009)	✓	○	○	○
This work	✓	✓	✓	✓

5.2.1 Selection of the Communication Protocols

FIPA’s *Agent Communication Language* (ACL) is a general reference specification for designing and implementing the communication interfaces between agents in a multi-agent system (Labrou et al., 1999). *ACL* defines the usage of *performatives*, which rule the intentions of the agents. Built-in performatives like *Propose*, *Accept*

Proposal, *Reject Proposal* and *Confirm* can be used in a negotiation scenario, but they are not adequate for an auction scenario. However, it is a reference specification from the agent community and thus included for the analytical evaluation.

The *Market Design Tournament* of the *Trading Agent Competition*, called *CAT*, defines its own communication protocol, which is called *CATP* (Niu et al., 2009). *CATP* allows the trading clients to be connected to the *CAT* server and trading data for the defined tournament scenario to be exchanged. *CATP* is the communication protocol that is currently applied in the *CAT* multi-agent system and therefore selected for analysis.

ClassAds is a job description language for matching job requests to machine descriptions used, used in the Condor's³ cluster management system (Liu and Foster, 2004). Languages like *Job Submission Description Language* (JSDL), *GLUE Schema*, *Resource Specification Language*, *Job Definition Language* and *Resource Specification Language* have been widely adopted as a technical resource and job description language in Grid projects like *Globus Toolkit* and *EGEE* (Andreetto et al., 2010; Laure et al., 2006; Smirnova, 2009). The *Common Information Model* (*CIM*) is a DMTF standard, which offers a detailed technical description of the information concepts and their relations in computing systems (DMTF, 2010b). The *Open Virtualization Format* (*OVF*) is a DMTF initiative for configuring and describing virtual machines, as well as the deployment procedure of the required applications. In contrast to *OVF*, the *OASIS Solution Deployment Descriptor* only focuses on the deployment and configuration procedure of software components (Galán et al., 2009). All of these languages are currently used for expressing technical requirements and configurations.

The *WS-Agreement* is a established “standard” in the Grid community for negotiating and establishing service level agreements (Andrieux et al., 2007). The *WS-Agreement* framework provides an abstract framework with core concepts for supporting the negotiations of key performance indicators, as well as price and penalty constraints, rather than concrete specifications on how to realize them in a real world setting. Moreover, the general framework of the *WS-Agreement* can be used as a wrapper to any of the previously discussed job description languages. Other communication protocols for negotiating service level agreements are the *Contract Net Protocol*, *WSLA*, *SLAng* and *RBSLA* (Smith, 1980; Paurobally et al., 2007; Paschke et al., 2005). However, *WS-Agreement* remains the current OGF proposal for service level agreements and has been generally applied in past and current research projects (Kübert et al., 2011).

³<http://www.cs.wisc.edu/condor/description.html>.

Nisan (2006) specifies a theoretical framework for bidding in combinatorial auctions. Consumers and providers can express combinatorial bids by using *OR*, *AND* and *XOR* operators on *Atomic Bids*. An *Atomic Bid* contains the number of items and valuation of the transaction object provided or demand.

Schnizler (2008) specified the *Multi-Attributive Combinatorial Exchange* language, which aims to define atomic bids not just for generic items, but also for a set of quality of service (QoS) and time attributes. QoS attributes can represent technical information like CPU, memory and storage. The time attributes specify the start and end times for executing a job, which are used to solve problems that arise in determining a winner for offline and combinatorial bids for a given set of consumer and provider bids.

Vilajosana et al. (2009) aim to specify a communication protocol, which is designed for a market-based scheduling domain and combinatorial auctions. However, the protocol defines its own technical attributes rather than reusing commonly accepted specifications like JSDL and GLUE. Furthermore, its specifications have not been evaluated.

5.2.2 Analytical Comparison to the Desiderata

Expressing Economic and Technical Preferences

D1 is partially supported by most communication protocols. *Partially* means, the protocol specifies either technical or economic attributes for the description of transaction objects (i.e., computing services), but not both of them. An ACL does not provide concrete support for auctions or concrete specification for technical attributes, but its general framework enables it to be applied as a wrapper to existing languages like JSDL. However, it requires extensions and additional rule specifications in order to be applied in auction scenarios. Communication protocols like CATP and Nisan (2006) specify economic attributes for bidding in tournaments and in combinatorial auctions. Therefore, CATP and Nisan (2006) cover only half of the desiderata and are designed for other domain scenarios, not for the scenario investigated in this work. *ClassAds*, *JSDL*, *CIM*, *OVF* and the other languages in the second cluster of Table 5.1 specify only the technical attributes for describing technical requirements of jobs or descriptions for the configuration and deployment of jobs to computing services. These communication protocols are good candidates to be wrapped into general frameworks with economic attributes with respect to the target adoption scenario. Only WS-Agreement, MACE and Vilajosana et al. (2009) fully satisfy D1. As a general framework for negotiations, a WS-Agreement can wrap any of the

technical description languages. However, a WS-Agreement is designed mainly for negotiations rather than for auctions. Moreover, as a general framework, the WS-Agreement includes attributes for structuring economic information like valuation, key performance indicators and penalties, however, it does not specify concrete instances or evaluate real use cases of them.

MACE supports the specification of both technical and economic attributes for computing services. As a framework, it can adopt technical attributes from technical description languages like JSDL and GLUE. Vilajosana et al. (2009) specify a communication protocol for the market-based scheduling of computing services. However, it is designed for combinatorial auctions and defines its own technical attributes, which are different from the commonly applied JSDL and GLUE specifications.

Separation of Private, Public and Market Messages

None of the communication protocols analyzed provide a model, which clearly separate the private, public and market messages submitted between the applications, computing services, agents and the market. Theoretically, each of the communication protocols can be extended to do this with the required additional attributes and rules, however, the different communication protocols are designed with their own philosophy and goals for their specific domain. Redesigning or modifying them with extensions will affect the completeness of the already defined attributes and original application model.

Support for the Creation of Service Contracts

Most communication protocols do not provide a well-defined support for the creation of service contracts as indicated in D3. The CATP protocol defines the notion of *transaction*, which identifies a match between two bids through the creation of a transaction id from the bids' ids (e.g., *Id: ask3, bid2*), as well as with the price statement (*Value: 90*). The CATP *transaction* is suitable for the CAT tournament scenario, but limited for the creation of service contracts in a market-based scheduling domain. MACE specifies a winner determination mechanism for combinatorial settings, however, a method for the creation and exchange of service contracts is neither suggested nor implemented. Only a WS-Agreement provides a well-defined support for negotiation and creation of a contract. However, the WS-Agreement only defines economic attributes with a negotiation-specific semantic. Auction-specific attributes like *bid*, *clearing price* and *limit time* are missing. A WS-Agreement does not include attributes for expressing payment information like prepayment, post-payment and account information. Moreover, a method for consumers and providers to sign WS-Agreement contracts to establish a legally binding agreement is not proposed.

Providing Clear Semantic Descriptions of Market Concepts

Most communication protocols do not provide explicit specification of an ontology, i.e. a formal semantic model of the utilized information concepts and their relations, as required in D4. Most of these protocols are modeled in an XML schema and the information concepts are specified as technical documentation, which is not in a machine-interpretable format unlike an ontology language like OWL⁴.

The ACL and CIM are the only communication protocols in Table 5.1, which explicitly support the definition of ontologies or are modeled as an ontology. The ACL language consists of concepts for defining and exchanging ontologies, as well as concepts for their logical validation. CIM is an extensive reference model for describing the information concepts and their relations in the domain of computing systems, such as their applications, devices, components and subsystems (DMTF, 2010b). Quirolgico et al. (2004) proposed a model to represent CIM as a formal ontology. However, these ontologies partially support the domain of this work and do not explicitly define the required market concepts, their relations and rules in a market-based scheduling exchange.

5.3 MX/CS: Message Exchange in Computing Service Markets

Figure 5.1 presents the overall communication architecture of *MX/CS* (cf. bidding scenario in Section 2.4.4).

Calls 1, 2 and 5 are invocation requests of the message types *PrivateMessage* (“request for bid”) and *PublicMessage* (bid), where 3 and 4 are the returned result (*MarketMessage*) of the invocation requests. *A* represents the invocation and response messages of the *Market Information Service*, which provides market information of the last N bids and clearing prices.

For instance, the *Application Orchestrator* instance invokes the interface method *bidCall(credential, privateMessage)* of *BidGenerator* in order to submit an application’s *privateMessage* together with the consumer’s credentials. The *BidGenerator* instantiates the selected bidding strategy, invokes the security component to receive a valid *samlAssertion* and a *digital signature* (cf. Section 4.4.6), generates a bid, signs and submits it (*privateMessage*) to the market. When there is a match, the

⁴Web Ontology Language.

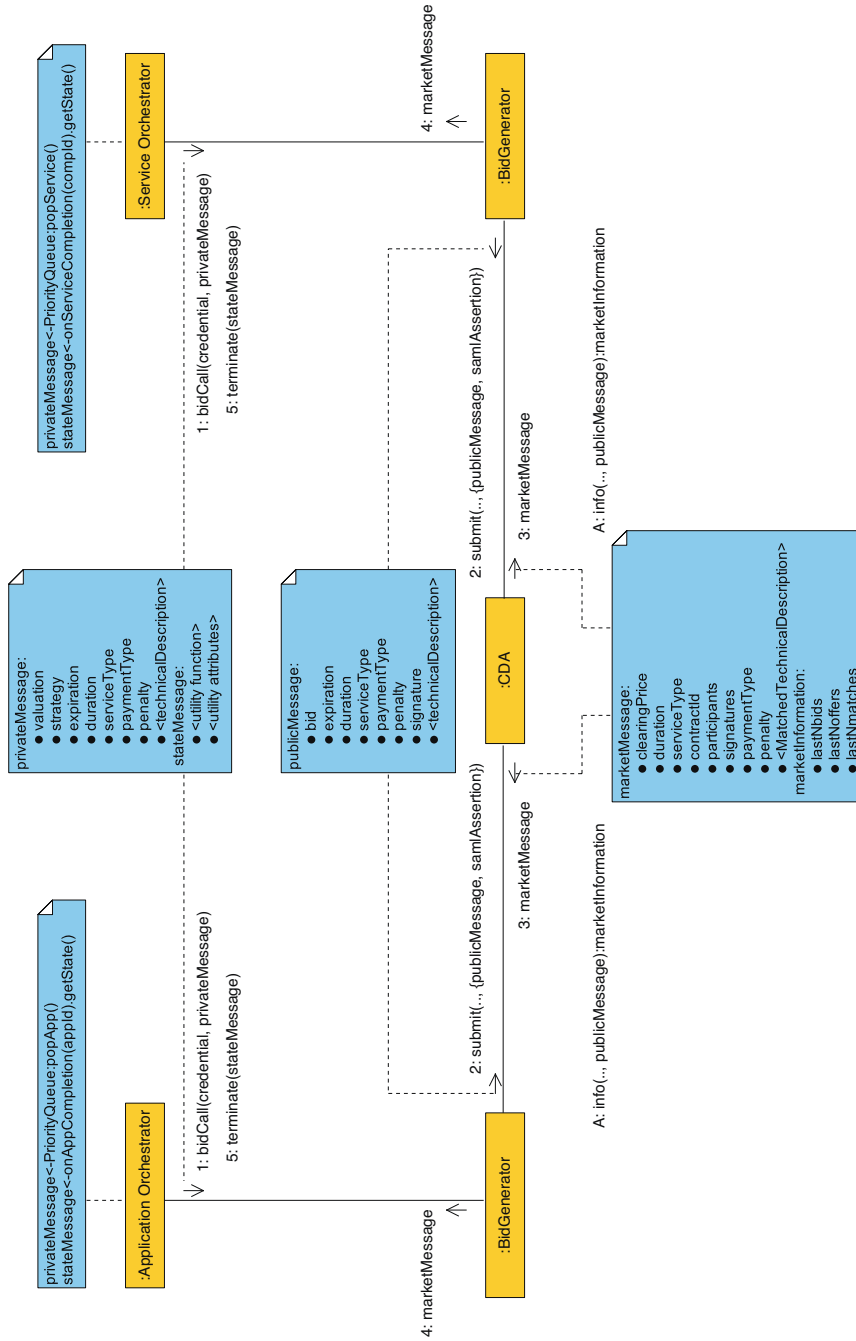


Figure 5.1: Communication protocols for the automated provisioning and usage of computing services

signed market contracts (*marketMessage*) are sent back to the *Application Orchestrator* through its *BidGenerator*. For each finished application, the *Application Orchestrator* sends the application's final monitoring data (*stateMessage*) using the *BidGenerator*'s *terminate* interface method. The data in *stateMessage* updates the knowledge base of *BidGenerator*, which is used by the implemented bidding strategies. These processes are similar for the provider side. The following sections specify the message types introduced in detail.

5.3.1 Private Message

According to Figure 5.1 (step 1), the consumer's *Application Orchestrator* and provider's *Service Orchestrator* submit a *PrivateMessage* to their own *BidGenerators*. The *PrivateMessage* models a scenario where consumers and providers do not reveal their true valuations and bidding strategies for the demanded or supplied transaction object. However, the consumers and providers trust their own *BidGenerator*, which implements agents and a pool of different bidding strategies. Therefore, a *PrivateMessage* of type \mathcal{P}_i for an *Application Orchestrator* or a *Service Orchestrator* i has the following structure:

$$\mathcal{P}_i = \{v_i, s_i, z_i, c_i, \Lambda_i, \Pi_i, \Delta_i, \Omega_i, U_i\} \quad (5.1)$$

v_i represents the consumer or provider valuation per unit of time (hour) for the requested or provided computing service. s_i is the identifier (class name) of the bidding strategy that is implemented within the *BidGenerator* framework. The s_i identifier allows the dynamic initialization of bidding strategies from the pool. z_i describes the expiration time of the bid in the order book. z_i plays a strategic role in bidding strategies for call markets where the market clearing is executed in time intervals, which are known to the agents (Gjerstad, 2003; Vytelingum et al., 2008; Schwartzman and Wellman, 2009). Therefore, z_i is obligatory for consumers and providers. c_i is the estimated maximum time, which is required for the consumer application (Ali et al., 2004). In the case of providers, c_i is the time (in hours) that a computing service is available on the market⁵. Λ_i specifies the type of computing service required,

⁵An initialization with *infinite* means that the computing service has no availability constraints and can be matched with any consumer bid. A time constraint limits the availability of the computing service. For example, a provider X knows that her/his cluster of 20 machines will be idle during the next 10 hours, but needed thereafter; and X decides to supply the 20 idle machines to the market for the next 10 hours. In this case, the provider bids are only matched to consumer

“BATCH” or “WEBSERVICE”. In the first case, the application is deployed to the computing service through a secured network connection (“Batch Supply Chain Data Analysis” scenario in Section 2.4.1.1). In the second case, the application is a Web service, which is deployed on an isolated web container environment like *Tomcat* or *Jetty* that runs on a dedicated computing service (“Interactive Sensor Data Analysis Scenario” in Section 2.4.1.2). The payment method Π_i indicates whether the payment should be executed “BEFORE” deployment of the application (prepayment), “AFTER” the execution of the application (post-payment) or “EITHER.” Π_i is obligatory for consumers and is set to “EITHER” as a default. Payment information is required by SORMA’s *Payment* component and enforced by *SLA Enforcement* as part of the *Contract Management* component (Nimis et al., 2009).

Δ_i specifies the name of the selected penalty function for calculating the compensation when a contract is not fulfilled. Becker et al. (2008) present a fair and incentive compatible penalty function for computing markets. Even though penalty functions are not part of this research, they are crucial for computing service markets since they offer an additional mechanism for establishing trust, reliability and acceptability for such types of markets.

Ω_i contains the *Technical Description* of the required or provided computing service. The specification of *Technical Description* is left open intentionally since the aim of *MX/CS* is to specify a bidding language for computing service markets by reusing common XML-based languages like *JSDL*, *GLUE* or *OVF* for technical descriptions, deployments and configurations. Finally, U_i contains the unique identification of the selected scoring function.

Valuation v_i , bidding strategy s_i and U_i are private attributes, which are known only to the owner’s *BidGenerator*. The remaining elements are transformed into a *PublicMessage*, which is the bid submitted to the market. Since this work does not include the research on penalty functions, the Δ_i element is optional, however, it was an important requirement in the SORMA scenarios. The Ω_i is implemented with a commonly applied computing service description language like *JSDL*. The access URI to the offered computing service is provided as part of the provider’s Ω_i and added to the contract on matches.

applications that do not violate the specified time constraint.

5.3.2 State Message

A *stateMessage* transfers the final state information about an executed application or finished computing service usage. This information is used as feedback for consumers' and providers' knowledge bases and utilized by their bidding strategies. It is composed of two main parts, the *ScoringFunction* U_i and related *Parameters*:

$$\mathcal{S}_i = (U_i, \{b_1, \dots, b_n\}) \quad (5.2)$$

In order to make scoring functions configurable, i.e., not hardcoded in the system, they are encoded in a *Lisp*-like syntax and stored in the *BidGenerator*'s knowledge base with an associated unique name. Rule engines like *JESS*⁶ or *Drools*⁷ are applied for loading and processing the scoring function U_i on runtimes with its associated attributes $(b_1 \dots b_k) \in B_i$, which are part of the overall attribute definition. The attributes and values for the scoring function are transferred in the *Parameters* element of the *stateMessage*.

$$\text{minCompletionTimeAndPayment}(v_a, F_{s,a}, \pi_{s,a}, c_a) = v_a F_{s,a} + \pi_{s,a} c_a \quad (5.3)$$

The Equation 5.3 presents an example of the scoring function called “Minimize Weighted Completion Time and Payments” or *minCompletionTimeAndPayment*, which is expressed below as a *Lisp*-like syntax:

```
1 (deffunction minCompletionTimeAndPayment(?valuation ?completionTime ?clearingPrice
   ?duration)
  (return (+ (* ?valuation ?completionTime) (* ?clearingPrice ?duration))))
```

The following snipped XML is an example of a transferred *stateMessage* for an executed batch application. The consumer's aim is to minimize the completion time and payments for her or his batch applications and therefore chooses the *minCompletionTimeAndPayment* scoring function to update *BidGenerator*'s knowledge base. The contract identifier is used to find the associated application and provider identifiers in *BidGenerator*'s knowledge base.

```
2 <StateMessage stateId='456 '>
  <ScoringFunction>minCompletionTimeAndPayment</ScoringFunction>
  <Parameters>
  4   <valuation sequenceNr='1'>35</valuation>
    <timeToComplete sequenceNr='2'>3</timeToComplete>
  6   <finalPrice sequenceNr='3'>25</finalPrice>
```

⁶<http://www.jessrules.com>.

⁷<http://www.jboss.org/drools>.

8

```
<duration sequenceNr='4'>5</duration>
</Parameters>
</StateMessage>
```

The associated key performance indicators $(b_1 \dots b_k) \in B_i$ are transferred with the *Parameters* element of *stateMessage* and used by the scoring function to calculate the reward of the application execution. The *timeToComplete* attribute indicates the time needed to complete the application execution and the *clearingPrice* attribute indicates the final price, which is probably different from the one in the contract due to compensation payments. The updated rewards are used from *BidGenerator*'s bidding strategies to adapt their bids to the supply and demand.

5.3.3 Public Message

Based on the *PrivateMessage* received, the *BidGenerator* instantiates the selected bidding strategy and generates a bid in the form of a *PublicMessage* \mathcal{B}_i (step 2 of Figure 5.1), which is submitted to the market. The *PublicMessage* is created from the *PrivateMessage* and contains the following transformed and additional elements:

$$\mathcal{B}_i = \{q_i, \tilde{z}_i, c_i, \Lambda_i, \Pi_i, \Delta_i, \Upsilon_i, \Omega_i\} \quad (5.4)$$

Most of the *PublicMessage* elements are transformed from the *PrivateMessage*. In *PublicMessage*, the *PrivateMessage*'s *bidding strategy* and *valuation* elements are replaced with q_i , which stores the value of the generated bid according to the selected bidding strategy. In *PublicMessage*, the expiration time $\tilde{z}_i \neq z_i$ is a dynamic parameter, which is adapted by some of the bidding strategies. If z_i was not specified in *PrivateMessage* and the selected bidding strategy does not update the value in \tilde{z}_i , then \tilde{z}_i is initialized with a default value (e.g., 24h). In addition to the *PrivateMessage*, the *PublicMessage* specifies the Υ_i element, which is the signature added by the security component in order to sign the bid and make it legally binding. The signature element is used by SORMA's *Contract Management* to sign the market matches between consumers and providers in the form of legal electronic contracts.

5.3.4 Market Message

The submission of the *MarketMessage* is depicted as step 3 of Figure 5.1. A *MarketMessage* is created as the result of a successful match between a consumer i and

a provider j bid. The *MarketMessage* has the following content:

$$\mathcal{X}_{i,j} = \{cId_{i,j}, \Sigma_{i,j}, \Upsilon_{i,j}, \pi_{i,j}, c_{i,j}, \Lambda_{i,j}, \Pi_{i,j}, \Delta_{i,j}, \tilde{\Omega}_{i,j}\} \quad (5.5)$$

$cId_{i,j}$ is the contract ID, which is uniquely created by the *Contract Management* of the consumer and provider identities, which are part of the $\Sigma_{i,j}$ element. The $\Upsilon_{i,j}$ element contains the consumer's and provider's signatures, which are taken from their *PublicMessages*. $\pi_{i,j}$ represents the calculated clearing price by the market mechanism. $c_{i,j}$ and $\Lambda_{i,j}$ are taken from the consumer's *PublicMessage*. $\Pi_{i,j}$ and $\Delta_{i,j}$ are taken from the provider's *PublicMessage* since they are values added to the provider's business strategy. $\tilde{\Omega}_{i,j}$ is the matched *Technical Description* with adjusted upper and lower bounds as a result of the matchmaking process according to the consumer's technical requirements and the provider's technical specification in their respective *PublicMessages*.

The *MarketMessage* describes the core concepts of a digital contract for a market-based scheduling domain. The *MarketMessage* is returned back to the *Application Orchestrator* or *Service Orchestrator* through the *BidGenerator* components. Subsequently, the Application Orchestrator deploys and executes the application on the allocated computing service according to the legally binding agreements in the *MarketMessage*. The URI to the provider's computing service is part of the *MarketMessage*'s $\Omega_{i,j}$.

In the SORMA context, the created *MarketMessage* was transformed into a *WS-Agreement* document since the *WS-Agreement* is an OGF specification for service level agreements (Andrieux et al., 2007; Borissov et al., 2009b). However, a *WS-Agreement* offers a reference framework for high-level service level agreement concepts rather than a concrete and detailed method for implementing them. Furthermore, a *WS-Agreement* was designed for negotiations rather than for auction scenarios. Borissov et al. (2009b) describe a mapping procedure to transform a *MarketMessage* into the existing concepts of the *WS-Agreement* specification. However, there are no semantically corresponding *WS-Agreement* elements for the *MarketMessage* concepts $q_{i,j}$, $c_{i,j}$, $\Lambda_{i,j}$, $\Pi_{i,j}$ and $\Upsilon_{i,j}$, which haven been *additionally modelled and extended* in the *WS-Agreement* framework (Borissov et al., 2009b). Furthermore, signatures $\Upsilon_{i,j}$ are an integrated part of the latest *PublicMessage* and *MarketMessage* definitions, however, the *WS-Agreement* specification does not contain an equivalent element. However, as a general and composite framework for negotiations, the *WS-Agreement*

integrates *WS-Security* elements to incorporate the signatures, but the link to *WS-Security* remains on an abstract level with respect to the clear security and trust methodology in the *PublicMessage* and *MarketMessage*.

5.3.5 Market Information Message

The *BidGenerator* has to know where to submit the bid generated, *PublicMessage*. The market platform *CSpace* integrates a *Market Information Service*, which enables queries for the set of available auctions and their market information (Nimis et al., 2009). The *info* method, which is part of *BidGenerator*'s agent interface, implements the functionality to request market information about available markets:

```
void info(string query, PublicMessage bid)
void handleInfoResponse(MarketInformation marketInfo)
```

Each of the available auctions in *CSpace* defines the scope of the traded transaction objects in terms of technical attributes for CPU, memory and storage with associated ranges, e.g., CDA auction for “middle” service configurations of 4 to 8 CPUs or CPU cores, 8 to 16 GB of memory and 500GB to 1TB storage. The following example returns the auctions and respective identifiers that match the technical description in *PublicMessage*:

```
1 <queries>
  <auctions/>
3 </queries>
```

The matchmaking algorithm is the same as the one used by the auction to match the bids, but is only applied to the technical attributes.

The following example queries market information about the “CDA” auction identifier. The query requests the last 3 consumer and provider bids, as well as the last 3 clearing prices from the market information service:

```
1 <queries>
  <auctions>
3   <auction id='CDA'>
      <marketinformation>
5         <lastNConsumerBids>3</lastNConsumerBids>
          <lastNProviderBids>3</lastNProviderBids>
7         <lastNClearingPrices>3</lastNClearingPrices>
      </marketinformation>
9   </auction>
  </auctions>
11 </queries>
```

The responses from the market information service are handled by the *BidGenerator*'s interface method *handleInfoResponse*. An example response of the previous query is presented in the following code snippet:

```
1 <response>
  <auctions>
3   <auction id='CDA'>
    <marketinformation>
5     <consumerbids><value>17</value><value>15</value><value>16</value></
      consumerbids>
    <providerbids><value>17</value><value>16</value><value>18</value></
      providerbids>
7     <clearingprices><value>17</value><value>16</value><value>15</value></
      clearingprices>
    </marketinformation>
9   </auction>
  </auctions>
11 </response>
```

5.4 Matchmaking of Public Messages

The allocation of a set of a priori known goods and services with varying characteristics to an a priori number of known requestors with varying preferences is called a *Winner Determination Problem* (WDP). WDPs are NP-complete since multi-attributive requestors' preferences are matched against multi-attributive service description characteristics (Nisan, 2006). Another characteristics of WDPs is that they often solve allocations of a priori known information in offline settings. Therefore, offline algorithms are not adequate candidates for a market-based scheduling context since WDPs are not scalable for thousands of applications and computing services. Online mechanisms like the *Continuous Double Auction* are efficient in matching supply and demand. In addition, they are scalable since consumer and provider bids are continuously matched with the bids available in the order book as soon as they arrive. In the context of market-based scheduling, the matchmaking process has to consider both the technical and economic preferences of consumers and providers. Moreover, in an online multi-agent scenario, the *matchmaking processes should be automated, efficient and fast* (Sycara et al., 2002). A survey of mechanisms for matching technical attributes is presented by Liu and Foster (2004). Liu and Foster (2004) discuss the *Condor ClassAds* language and matchmaking mechanism in more detail and propose a novel matchmaking mechanism, called *Redline*. *Redline* allows the specification and matching of value ranges like "CPU tact frequency between 2 GHz and 3 GHz." JSDL is also specified to support the value ranges for technical attributes (Anjomshoaa et al., 2005). However, there is no known research that describes the

automation of the matchmaking processes of both the technical and economic preferences of consumers and providers.

Sycara et al. (2002) summarize three general types of matchmaking mechanisms called *Exact match*, *Plug-in match* and *Relaxed match*. *Exact match* describes a situation in which the requests of the demand and supply agents have equivalent variables and values. The *Plug-in match* is less restrictive, but offers a greater likelihood of finding a match rather than an exact match. The idea behind the *Plug-in match* is that the provider and consumer messages can differ in the description details of their respective technical attributes. For example, a provider’s technical description of the offered computing service may be very detailed in terms of the number of technical parameters. Conversely, a consumer’s technical description may utilize few technical parameters. According to the *Plug-in match*, both descriptions can match if the few technical attributes of the consumer description match part of the related provider’s description. The additional technical attributes may complement the matched document. A *Relaxed match* is the least restrictive matchmaking method. A *Relaxed match* does not return a match or no-match value, but determines the value distances of the different consumer and provider descriptions. A simple heuristic in this case could be that a consumer and provider description match if the overall “distance value is smaller than a preset threshold value” (Sycara et al., 2002).

In the context of SORMA, a kind of *Plug-in match* mechanism was implemented (Nimis et al., 2009). The matchmaking process of *PublicMessages* is performed by the implemented auction and matchmaking mechanisms in two steps – i) technical matchmaking and ii) price-based matchmaking. The technical matchmaking mechanism applies the characteristics of the selected technical description language, JSDL. JSDL is a well-applied description language in *Cluster* and *Grid* schedulers, as well as in many other related projects (McGough and Savva, 2008). As such, JSDL supports value ranges for technical attributes. Thus, the technical matchmaking mechanism is interfaced with the auction mechanism and implemented to support value ranges for the technical attribute-value pairs. The price-based matchmaking mechanism is performed by the auction mechanism itself. In the case of *Continuous Double Auction*, the bids are matched on arrival according to the *K-Pricing* schema, $price = kq_i + (1-k)q_j$ with $k = 0.5$ (Satterthwaite and Williams, 1989). The technical and economic matchmaking procedure is depicted in Algorithm 5.4.1 and described as follows.

Algorithm 5.4.1: MATCHMAKING(*publicMessage*)

comment: Price-based and technical matchmaking of bids and offers

```

bids ← SortedList(BidComparator)
offers ← SortedList(OfferComparator)
procedure MATCH(publicMessage)
  removeExpired()
  if publicMessage.isBid()
    for each offer in offers
      do if isTechnicalMatch(publicMessage, offer)
        then {
          if publicMessage.getBid() ≥ offer.getBid()
            then {
              then {
                deal ← createMatch(publicMessage, offer)
                remove(offer)
                break
              }
            }
          else if publicMessage.isOffer()
            for each bid in bids
              do if isTechnicalMatch(publicMessage, bid)
                then {
                  if publicMessage.getBid() ≤ bid.getBid()
                    then {
                      deal ← createMatch(publicMessage, bid)
                      remove(bid)
                      break
                    }
                }
        }
  if deal ≠ nil
    then {
      marketMessage ← deal.getMarketMessage()
      informAgents(marketMessage)
    }
  else if deal = nil
    then {
      insert(publicMessage)
    }

```

Step 1 – Technical Matchmaking: As a consumer bid or provider offer arrives in the form of a *PublicMessage*, the *match* procedure performs a technical matchmaking against a list of provider offers or consumer bids. Bids are sorted in descending order, and offers in ascending order. The technical description of the service is part of the offer’s and bid’s *PublicMessages*. The *isTechnicalMatch* method checks to see if the bid and offer messages are *technically compatible*. A consumer *i*’s *PublicMessage* and a provider *j*’s *PublicMessage* pair are *technically compatible* if their key technical attributes (Key Performance Indicators or KPI), which are part of the *PublicMessage*, match into non-empty values:

$$\text{for all } kpi_i \in KPI_i, kpi_j \in KPI_j, kpi_i \cap kpi_j \neq \emptyset \quad (5.6)$$

For example, the intersection of the consumer’s KPI $CPU\text{Speed}_i = \{2GHz, 3GHz\}$ ⁸, and the intersection of the provider’s $CPU\text{Speed}_j = \{2.6GHz\}$ amounts to $\{2.6GHz\}$.

⁸Here the *CPU Speed* is defined for a range of 2 to 3 GHz.

Step 2 – Economic Matchmaking: If a bid and an offer are technically compatible (the method *isTechnicalMatch* returns *true*), then the *match* procedure performs a price-based matchmaking. Since the bids and offers are ordered according to their values, a full match can be immediately returned if the consumer’s bid is higher than the provider’s bid, otherwise, the algorithm moves to *Step 1* with the next possible candidate (bid or offer) in the order book. If the consumer’s bid is less than the provider’s offer, then the matchmaking process can be terminated and the consumer’s bid remains in the order book until it is matched or the specified *expiration time* in *PublicMessage*. When a match is identified, the auction creates a *MarketMessage* and delivers it to SORMA’s *Contract Management*, which generates a binding contract and returns it back to the related consumer and provider. The technical attribute-value pairs of the *MarketMessage* are constructed from the intersection values of the consumer’s and provider’s *PublicMessages*. The *clearing price* is the result of the economic matchmaking and is calculated according to the *K-Pricing* schema.

5.5 The MX/CS Ontology

Software engineering and ontologies are complementary technologies to describe and share domain knowledge, which is “stored” in (complex) software systems (Gasević et al., 2009). The application of ontologies in the software engineering processes provides a common language for software engineers of the system concepts developed, as well as a common understanding of system’s requirements, their solution concepts, as well as the implementation details. Machine-processable system ontologies can help software engineers establish links between the system knowledge and software artifacts produced in order to automate the verification and integration processes of existing and novel components. Gasević et al. (2009) propose the use of ontologies for software engineering processes. For example, their application starts in the *requirement analysis* phase in the definition of a common *lexicon* for the targeted system, which serves to establish a common and well-defined vocabulary of the concepts, their properties and internal/external relationships for the people involved in the project. In the *design phase*, ontologies can be applied by software engineers to iteratively validate and reason their UML or software models. Furthermore, ontologies can facilitate the transformation and integration processes of third-party or novel components. In the software *implementation phase*, ontologies can be applied to document implementation decisions, i.e., selecting software patterns for a specific domain problem (Gasević et al., 2009). A formal description of the semantics of market concepts adopted in communication protocols is crucial to make sure that

they understand each other, as well as to ensure their interoperability, verification, and integration in existing and new components. This is especially important for interdisciplinary research in the area of multi-agent systems and market mechanisms.

Therefore, this research proposes separating the data exchange model from the semantic model. The data exchange model of *MX/CS* is specified as XML schema (Appendix C.1), the semantic model is specified in OWL (version 2.0) and called *MX/CS Ontology* (Appendix C.2). The *MX/CS XML schema* explicitly defines the data model of the presented message types *PrivateMessage*, *PublicMessage* and *MarketMessage*. The *StateMessage* and *MarketInformation* are specific to the *BidGenerator* and not directly involved in the communication between the clients (*Application Orchestrator*, *Service Orchestrator*) and the SORMA market components. Therefore, *StateMessage* and *MarketInformation* are specified externally to SORMA's market communication protocol (Appendix C.3), but related to the *MX/CS Ontology* (Appendix C.2).

Following is an example of an XML schema element definition. The defined element *strategy* of the type *string* stores the class name of the preferred bidding strategy:

```
1 <xsd:element name='strategy' type='xsd:string' sawsdl:modelReference='mxcsProtocol#
   strategy' />
```

SAWSDL enables annotating XML elements with a reference to an existing semantic concept that is part of an (domain) ontology. In the above example, the SAWSDL attribute *sawsdl:modelReference* provides a reference to the *strategy* concept in *mxcsProtocol.owl* (*MX/CS Ontology*).

In the following OWL snippet, the definition of the concept *strategy* is presented, which is part of the *MX/CS Ontology*:

```
1 <DataPropertyDomain>
   <DataProperty URI="\&mxcsProtocol;strategy" />
3   <Class URI="\&mxcsProtocol;PrivateMessage" />
   </DataPropertyDomain>
5   <DataPropertyRange>
   <DataProperty URI="\&mxcsProtocol;strategy" />
7   <Datatype URI="\&xsd:string" />
   </DataPropertyRange>
9   <EntityAnnotation>
   <DataProperty URI="\&mxcsProtocol;strategy" />
11  <Annotation annotationURI="\&dc:relation">
   <Constant>http://sw.opencyc.org/concept/Mx4rvViB_5wpEbGdrcN5Y29ycA</
   Constant>
13  </Annotation>
   </EntityAnnotation>
15  <EntityAnnotation>
   <DataProperty URI="\&mxcsProtocol;strategy" />
17  <Annotation annotationURI="\&rdfs:comment">
```

```

19         <Constant>A bidding strategy is a complete plan of actions for whatever
20           situation might arise; this fully determines the agent behavior. A
21           bidding strategy will determine the action of the agent will take at
22           any stage of the bid generation and market-based scheduling
23           processes, for every possible history and available market
24           information to that stage.</Constant>
25     </Annotation>
26 </EntityAnnotation>
27 <EntityAnnotation>
28   <DataProperty URI="\&mxcsProtocol;strategy" />
29   <Annotation annotationURI="\&rdfs;seeAlso">
30     <Constant>http://en.wikipedia.org/wiki/Strategy_(game_theory)</Constant>
31   </Annotation>
32 </EntityAnnotation>
33 <EntityAnnotation>
34   <DataProperty URI="\&mxcsProtocol;strategy" />
35   <Annotation annotationURI="\&rdfs;seeAlso">
36     <Constant>http://mitpress.mit.edu/books/FLAOH/cbnhtml/glossary-S.html</
37       Constant>
38   </Annotation>
39 </EntityAnnotation>
40 <Declaration>
41   <DataProperty URI="\&mxcsProtocol;strategy" />
42 </Declaration>

```

A concept is defined through its *Domain space* (here *strategy* is part of *PrivateMessage*) and its *Range space* (here *xsd:string*). Furthermore, the definition of *strategy* is associated with additional annotations like *comments*, *see also* links, as well as *relations*. A best practice in the ontology design is to associate ontology concepts to a well-known upper ontology, such as *OpenCyc* (Mascardi et al., 2007). The *MX/CS Ontology* is associated with concepts of the *OpenCyc* upper ontology by using the OWL type of *relation* annotation.

5.6 Extensions for Combinatorial Bids

Combinatorial markets and related combinatorial bidding languages have been extensively explored in the literature (Nisan, 2006; Schnizler, 2008; Lubin et al., 2008). A combinatorial bid is a logical combination of atomic bids, expressed through the logical operators *XOR*, *AND* and *OR*. Following example demonstrates an extension of the introduced message protocols (*PrivateMessage* and *PublicMessage*), which allows the specification of combinatorial bids of a set of *TechnicalDescriptions*, $\{TD_1, \dots, TD_n\}$, which are wrapped into the *PrivateMessage* and *PublicMessage* documents.

```
1 <Combinations>
  <XOR>
3   <AND>
4     <Service ref='TD1' />
5     <Service ref='TD2' />
6   </AND>
7   <Service ref='TD3' />
8 </XOR>
9 </Combinations>
```

In the case of a *PrivateMessage* definition, consumers specify their valuation for a service configuration or a set of alternatives (Borissov et al., 2008a). Suppose that the deployment, application execution and monitoring interfaces for computing services are standardized and adopted by all providers. Furthermore, suppose that *TD1* is a technical description of an Amazon EC2 service and *TD2* is a technical description of an Amazon S3 service. In contrast to *TD1* and *TD2*, *TD3* is a description of an integrated Google AppEngine service, which offers both – computational and storage facilities. In this example, a consumer specifies an exclusive-or *combinatorial bid* for either the *TD1* and *TD2* or *TD3* services.

5.7 Summary

This chapter presented the design and realization of the *MX/CS* communication protocol for a market-based scheduling domain. The derived design desiderata required that communication protocols for market-based scheduling must allow the expression of both technical and economic attributes; it must support different types of message for private, public, contract and market information; it must enable the creation of legally binding service contracts; and it must contain semantic descriptions of the implemented domain concepts. The analysis of existing communication protocols showed that there is no existing communication protocol, which satisfies the design desiderata.

The *MX/CS* communication protocol specifies the economic attributes needed to implement a market-based scheduling scenario. Furthermore, as a framework, *MX/CS* is modeled to wrap any technical description language for computing services. Therefore, desideratum D1 is fully satisfied.

Desideratum D2 is fully supported with the specification of the various message types: *PrivateMessage*, *PublicMessage*, *MarketMessage*, *StateMessage* and *Market-Information*. This specification is consistent with a realistic bidding scenario in which consumers and providers have full control of their own bidding infrastructure and

implemented bidding strategies. The *PrivateMessage* type models the exchange of internal information within consumers' and providers' private environments. The *PublicMessage* models the bid that is sent to the auction. *MarketMessage* is the contract created when there is an auction match, which is returned from the auction to the allocated consumer and provider. The *StateMessage* and *MarketInformation* types are used from *BidGenerator* to update its knowledge base and request market information.

The creation of electronic contracts between consumer and provider bids is supported with the specified concepts in *MarketMessage* (D3). A contract (*MarketMessage*) between a consumer and a provider bids (*PublicMessages*) is created if they match technically and economically. The *MarketMessage* contains the digital signatures of the participants and is sent back to them in the form of a *WS-Agreement* document, which contains the auction-specific concepts from the *MarketMessage*. Moreover, the XML data model of *MX/CS* is associated with the *MX/CS* ontology, which explicitly defines the utilized concepts in human and machine-readable form (D4). The *MX/CS* ontology captures a part of the SORMA system knowledge for the associated system components and their interactions. With respect to the best practices in ontology design, the *MX/CS* ontology concepts are linked to the well-known upper ontology *OpenCyc*.

The *MX/CS* has been implemented as a proof-of-concept in the SORMA prototype. In the Cloud context, the *MX/CS* communication protocol can be applied as a wrapper to any technical description language for infrastructure services (IaaS) in order to support the exchange in a market-based scheduling scenario. A detailed description of the *MX/CS* integration in three case studies, as well as in a *Social Cloud* scenario is presented in Chapter 7.

Part III
Evaluation

Chapter 6

Agent-Based Numerical Experiments

THIS chapter presents an economic evaluation of *Q-Strategy* for consumers and providers in competition settings against benchmark strategies in spot markets. Section 6.1 depicts the evaluation methodology in detail followed by the setup of an agent-based environment. Section 6.2 presents the results of the agent-based experiments and Section 6.3 summarizes the chapter.

6.1 Evaluation Methodology

Evaluation of market mechanisms and bidding strategies for realistic settings is a complex task dealing with the high heterogeneity of agents, varying preferences and incomplete information. The performance of an agent depends not only on its own preferences, but also on the strategies of other agents, and fluctuating supply and demand. The resulting evaluation space becomes large with the number of possible agent strategies, nevertheless it is necessary to verify agents' performances under these changing conditions (Sodomka et al., 2007).

The newly developed bidding strategy, *Q-Strategy*, is evaluated against benchmark strategies in homogeneous and heterogeneous settings. The bidding strategies are instantiated into bidding agents, according to the defined evaluation scenario. The agents generate the bids automatically on demand – consumer agents upon receipt of a new job; provider agents, each time a computing service becomes idle. In order to reduce the technical bias through network communication and concentrate on the economic outcome, all experiments are executed in a controlled multi-agent environment with a market mechanism, and a corresponding configuration of consumer and provider bidding agents running on a single machine instance. The design of the experiments and the evaluation methodology is in keeping with the proposed methodologies for multi-agent numerical experimentation postulated in *Agent-based Computational Economics* and *Empirical Game-theoretic Analysis* (Tesfatsion, 2006; MacKie-Mason and Wellman, 2006) (see also Section 2.5).

Table 6.1: Design of the experiments and variables

Variables	Values
<i>Number of provider agents</i>	50, 100
<i>Provider settings</i>	Homogeneous agents of {ZIP, GD, Q-Strategy}
<i>Number of consumer agents</i>	10 in total
<i>Consumer settings</i>	homogeneous and heterogeneous scenarios Q-Strategy vs. Benchmark ∈ {ZIP, GD} in ratios of (0:10), (1:9), ..., (10:0) agents
<i>Number of consumer applications</i>	Real job profiles (Feitelson, 2010): LLNL* with ca. 13000 job profiles HPC2N* with ca. 110000 job profiles
<i>Job duration c_j</i>	$c_j \in [1, 24]$ hours
<i>Consumer's valuation v_j per hour</i>	$N(190, 25)$, $v_j \in [70, 300]$
<i>Provider's valuation v_i per hour</i>	$N(75, 1.7)$, $v_i \in [70, 80]$
<i>Payment</i>	$\pi_{i,j}(v_j, c_j) = v_j c_j$
<i>Job's j completion time</i>	$F_j = t_{j,startBidding4App} - t_{j,finishAppExec}$
<i>Metrics</i>	Consumer's scoring function: $\max(U_j)$ with $U_j = -v_j F_j - \pi_{i,j}$ (min. opportunity costs and payments) Provider's scoring function: $\max(U_i)$ with $U_i = \pi_{i,j} - v_i c_j$ (maximize profit) Combined consumer and provider scores: $W = \sum_j U_j + \sum_i U_i$

Table 6.1 presents the global specification of the variables, the combinations of which define the overall scope of the experimental settings. The details are discussed in the following sections.

6.1.1 Setup of the Market Mechanism and the Application Data Profiles

The target market mechanism for all experiments is a spot market by means of the online *Continuous Double Auction*. For each experiment setting, the same two real data profiles – LLNL and HPC2N – were executed separately and independently from each other in the different settings. The LLNL and the HPC2N application data

profiles¹ i) offer stable characteristics with small fluctuations of the jobs' technical parameter values and ii) these workloads have also been applied in other research works (Feitelson, 2010). Originally, the LPC EGEE was also selected as a third candidate, but the experiments in Borissov et al. (2010) showed that the evidence from the outcome with the HPC2N workload is similar to that of the LPC EGEE workload. Therefore, the LPC EGEE workload was skipped from the evaluation scenarios in order to save two months of computing time for additional 1200 jobs, which did not result in a significant difference in the outcome other than of the HPC2N workload.

The job profile characteristics in the LLNL and HPC2N workloads are in keeping with the scenarios selected for this work (Section 2.4.1), where consumers execute the same applications for demand forecasting and image processing, but with slight variations in job duration (e.g., between 1 and 8 hours in the case of TXTDemand, Section 2.4.1.1) based on the input data for the applications. Moreover, the log files have been filtered to support application types with durations between one hour and twenty-four hours. The lower bound is motivated by the fact that current Cloud IaaS offerings are for at least one hour of usage; the upper bound is motivated by the investigated application case studies in Section 2.4.1, which define short-time batch applications (overnight execution with a maximum of 8 hours), as well as interactive and quasi real-time applications for image processing with a minimum of one hour usage time in total. It is assumed that usage time of less than one hour is performed on internal (own) computing infrastructures. In times of high demand, however, it is assumed that consumers purchase external computing services for at least one hour (cf. Section 2.4.1.2).

Moreover, the providers' minimum hour purchase is motivated by the fact that applications implicate time and bandwidth costs for reserving computing services, deployment, runtime and monitoring; and these costs may vary with application size. Therefore, it is not economically feasible to execute applications on outsourced computing services runtimes that are less than one hour and for providers to receive payment only for the net execution of the application without considering the total costs.

¹Feitelson's workload archive homepage: <http://www.cs.huji.ac.il/labs/parallel/workload/logs.htm>.

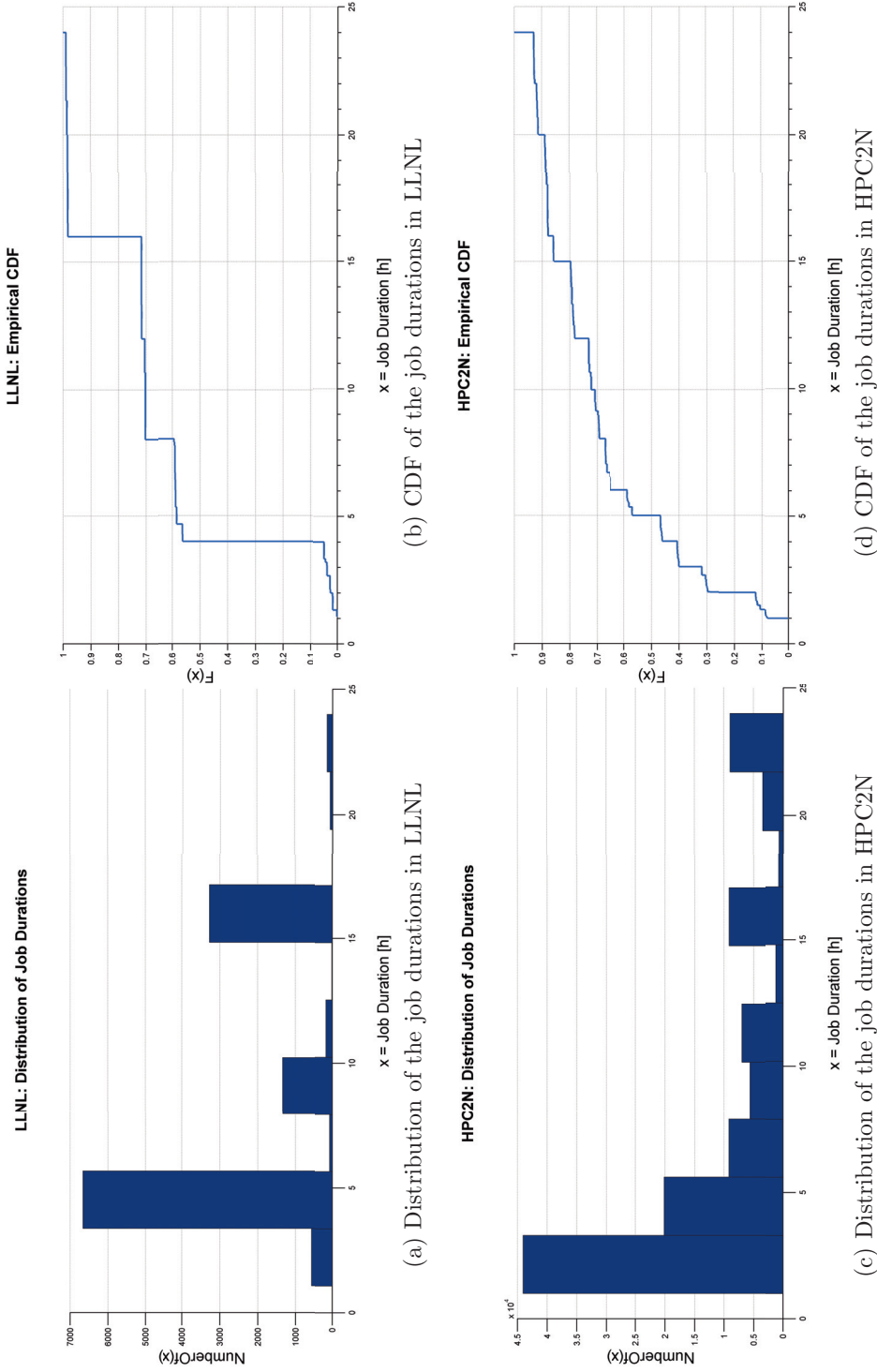


Figure 6.1: Distribution of the job durations and cumulative distribution function (CDF) of the data profiles

Figure 6.1 shows the distribution of the job durations and the cumulative distribution function of both data profiles. The job durations of the filtered job data profiles are in keeping with the characteristics of both case studies in Section 2.4.1. The LLNL data profile has a small differentiation of job duration types with the highest number of jobs that are 4 and 16 hours long. The mean job duration in LLNL is 8.01h and the standard deviation is 5.58h. The larger HPC2N data profile has a higher differentiation of job durations with a mean of 7.50h and a standard deviation of 6.96h. In HPC2N the highest number of jobs have a duration of 2 and 4 hours.

6.1.2 Setup of Bidding Strategies

The evaluation of this work focused on the consumer side in settings with varying homogeneous and heterogeneous agents, by applying the *Q-Strategy*, as well as the ZIP and GD strategies. In order to measure consumer outcome in a market-based scheduling scenario, a more feasible scoring function is needed than just profit maximization since application owners are not just interested in the price, but in the time-efficient execution of their applications (Buyya et al., 2002). Therefore, the analysis of the consumer outcomes was performed with the proposed scoring function in Heydenreich et al. (2010), namely the “minimize completion time and payments”. In order to reduce evaluation and outcome analysis complexity, the evaluation focused on the consumer side, where the providers are typically assumed to be profit maximizing agents. This focus solves two issues for the analysis.

Firstly, real-world providers of computing services apply more complex optimizations than just profit maximization. Complex optimizations in this case mean more sophisticated scoring functions like “maximize computing service utilization and profit,” “maximize consumer happiness, computing service utilization and profit” and many others. These optimizations are reflected in the provider’s business models, such as introducing price differentiations for different consumer types (e.g., “gold,” “silver” and “bronze”), sophisticated service level agreements, scheduling and utilization policies (Püschel et al., 2007; Becker et al., 2008). The analysis of the provider side and derivation of appropriate scoring function will deflect the focus of this work and it is an important part of many future doctoral theses. *Secondly*, assuming providers to be profit maximizers allows provider outcomes to be compared to the outcomes in the state-of-the-art literature.

The analysis of existing bidding strategies for the *Continuous Double Auction* and its variants showed that the GD and ZIP strategies are suitable and fair benchmark candidates for evaluation of the *Q-Strategy* and for the selected domain-specific market

of this work (Section 3.3.5).

Similar to existing works in the literature for agent-based evaluation, the selected total number of consumer agents amounts to 10 (Schvartzman and Wellman, 2009; Vytelingum et al., 2008; Tesauro and Bredin, 2002). Moreover, Tesauro and Bredin (2002) performed a kind of heterogeneous analysis of competing bidding strategies – GDX vs. ZIP, as well as GDX vs. GD types of agents, with “typical agent populations consisting of 10 buyer agents and 10 seller agents.” Table 6.1 shows the design of the homogeneous and heterogeneous settings. A homogeneous setting is given when all of the consumer and provider agents apply the same bidding strategy, otherwise the setting is heterogeneous. In heterogeneous settings, the *Q-Strategy* competes against one of the benchmark strategies, (*Q – Strategy : Benchmark*), in a ratio of (9 : 1) to (1 : 9) different agents.

In the three workload data profiles – LLNL, HPC2N and LPC EGEE, the number of computing nodes was around 50 and 100. Therefore, the number of provider agents in the investigated setting was selected to be either 50 or 100. The scenarios with 50 and 100 providers allows the outcomes to be compared from both a competitive and less competitive perspective. This differentiation allows the outcome to be evaluated under a different competition strength. Each provider agent represents a machine and executes an application exclusively in each time unit immediately after a market allocation for the specified application duration. The representation of one agent per machine is motivated by a competitive market-based scheduling scenario where each provider aims to maximize its goal. For example, Amazon offers computing services on a spot market, but it is the single provider for EC2 instances, rather than the double-sided case assumed for multiple providers and multiple consumers. Given that the focus is on the consumer, all of the provider agents apply only one type of bidding strategy, either the *Q-Strategy*, ZIP strategy or GD strategy. Thus, in each setting the provider agents are “homogeneous.”

The application data profiles do not contain information about the application or computing service valuations. The distributions that are usually selected for valuations are the uniform and normal distributions (Sandholm et al., 2008; Jiang and Leyton-Brown, 2007; Robu and Poutre, 2009). The normal distribution is commonly adopted for evaluating adaptive algorithms and thus selected for the generation of the application’s valuations (Jiang and Leyton-Brown, 2007). The selected distribution and range for the job valuations $\mathcal{N}(190, 25)$ are consistent with those of prominent authors on the subject of the agent-based evaluation domain for algorithmic bidding strategies (Schvartzman and Wellman, 2009; Tesauro and Bredin, 2002). The

generated job valuations (also called limit prices by some authors) are in the range $v_j \in [70, 300]$ and do not vary during the experiments; only the bids vary according to the selected bidding strategy. As with the providers, the computing service valuations are also normally distributed $\mathcal{N}(75, 1.7)$. The application scenario defines the provisioning of substitutable computing services with similar performance characteristics through well-defined interfaces and different goal maximizing providers. Therefore, the providers' i valuations are closer to each other and generated within a tighter range $v_i \in [70, 80]$ the valuations that of the consumers. The providers' valuations do not change during the experiments; they change according to their bids and the selected bidding strategy.

6.1.3 Setup of the Multi-Agent System

The multi-agent environment for the experiments integrates the *BidGenerator* (Section 4.4) and the market platform *CSpace* with a *Discrete Event Simulation Engine* (DES) (Borissov et al., 2008b; Nimis et al., 2009). DES is an integral part of a multi-agent experimental system since it controls the timing and distribution of events and their correct execution in a controlled environment (Chevaleyre et al., 2006; Ross, 2006). The DES engine uniformly distributes the request events of arriving applications from the target workload profile (LLNL or HPC2N) to the consumer agents. Each job from the workload profile is added as an event in DES, which is *run* and executed with the respective duration (*delay*) as specified in the workload profile.

The experimental environment communicates with the market platform *CSpace* by implementing the *ProtocolInWords* and *ProtocolOutWords* interfaces. In order to reduce unnecessary overhead (here the bidding strategies are evaluated; an evaluation of the system integration and performance is part of Chapter 7), the communication methods are directly invoked with the required attribute-value pairs for the experiments, instead of creating, serializing and parsing XML messages like in a real system (Section 5.3). The DES Engine runs the CDA protocol, which receives the messages from the agents with the *ProtocolInWords* interface, performs the economic match-making and responds to the agents with the *ProtocolOutWords*. The agent uses the *ProtocolInWords* interface to register with the market and exchange consumer *bids* and provider *offers*. The *ProtocolOutWords* interface is used by the CDA i) to send the *match* messages to the allocated consumer and provider agents, as well as to ii) *info* to broadcast (public information) other agents' actions, such as bids and clearing prices, to all registered agents.

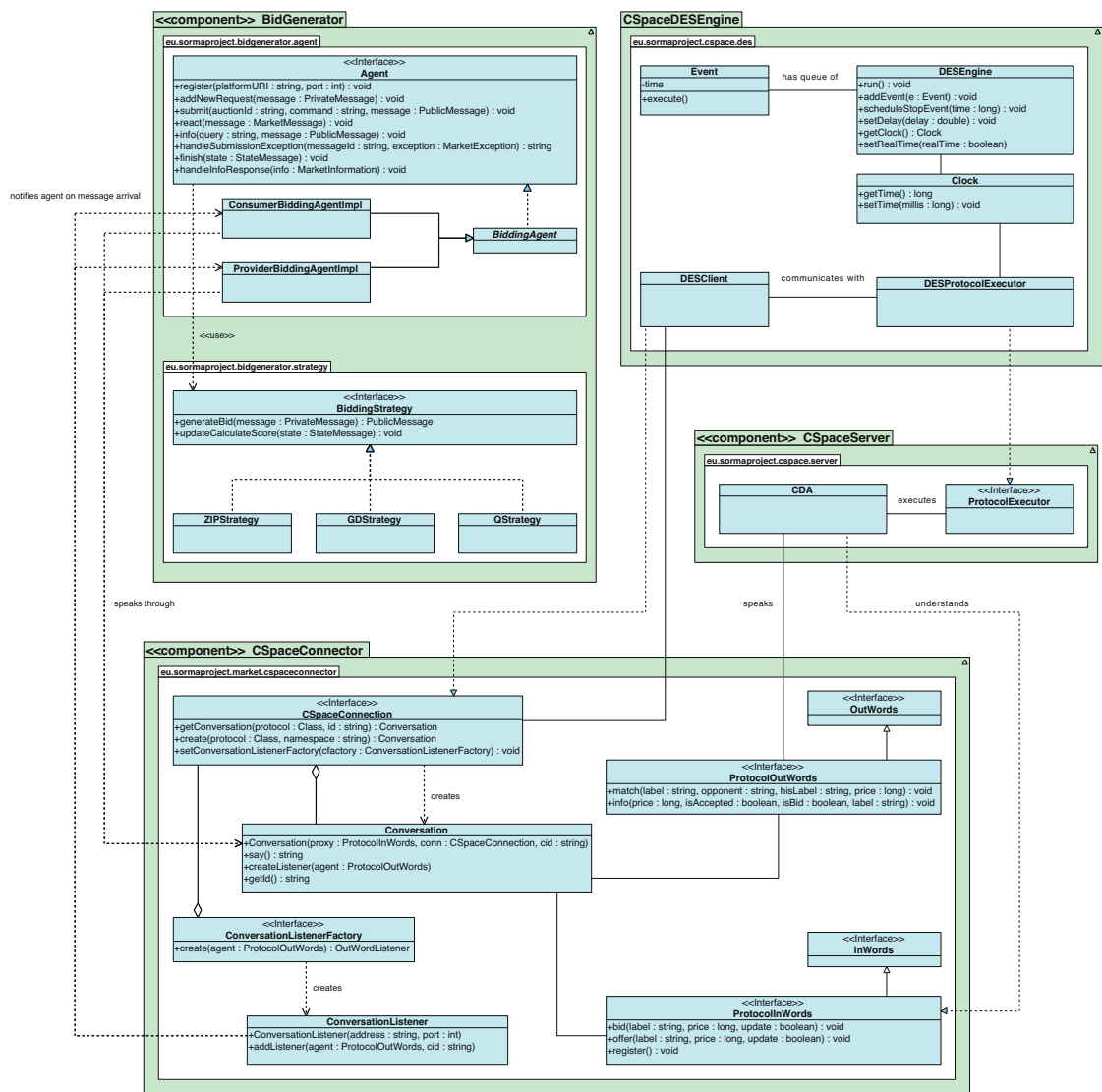


Figure 6.2: Architecture of an agent-based experimental environment with a Discrete Event Service Engine

The specified variables in Table 6.1 define the overall evaluation scenario, the combinations of which resulted in 240 unique experimental settings, where each one was repeated 10 times. The 2400 numerical experiments have been executed on 88 computing nodes over a period of four months on the HP XC3000 cluster system at the *Steinbuch Centre for Computing*².

6.1.4 Analyzing the Efficiency of Bidding Strategies

Common metrics in the economic literature to measure the efficiency of market mechanisms and bidding strategies are the overall score of consumers or providers and the welfare, and combination of both. The literature on automated bidding agents in CDA investigates the outcome of bidding strategies based on the classic assumption of *profit maximization*. In the field of market-based scheduling, it is assumed that consumers not only minimize payments (increase profit), but also to try minimize opportunity costs. Heydenreich et al. (2010) proposed a consumer scoring function for market-based scheduling, which aims to minimize both, the *weighted completion time* $v_j F_j$ and the *payments* $\pi_{i,j}$. Although Heydenreich et al. (2010) propose this scoring function for a strategy-proof decentralized mechanism in which consumers can act strategically and providers offer their computing services for free, this scoring function is a more appropriate metric outcome analysis and for the market-based scheduling domain rather than just the *profit maximization* score.

In Heydenreich et al. (2010), the v_j is defined as the costs for waiting for an additional time unit in the machine queue for execution, where F is the time between the “arrival” of the application from the consumer agent until the actual time of application completion. Within the scope of this work, v_j is simply defined as the valuation of the jobs – a higher valuation of a job represents higher opportunity costs for waiting for an additional time unit to be allocated on a machine. $\pi_{i,j}$ is the price per unit of time that a consumer j has to be pay to the allocated provider i for the required runtime c_j of the job.

In contrast to the mechanism in Heydenreich et al. (2010), providers in the *Continuous Double Auction* also behave strategically. Within the scope of this work, each of the providers are assumed to be traditional profit maximizing agents. With the selection of this traditional scoring function, a straightforward comparison of the providers’ outcome to the works of Tesauro and Bredin (2002) and Vytelingum et al. (2008) is possible to some degree.

²<http://www.scc.kit.edu>.

Finally, the aggregate consumer and provider scores represents the welfare (here referred to as *combined consumer and provider scores*) of the outcome, which is compared within the different settings.

6.1.5 Sensitivity Analysis

In order to initialize the variables of the parametrized bidding strategies – *Q-Strategy*, ZIP and GD – a *one-at-a-time* sensitivity analysis was performed (Saltelli et al., 2008). In a *one-at-a-time* sensitivity analysis, one parameter is varied in each of the sensitivity analysis experiments. The variation scope of the parameters was specified according to the specifications in the literature and calibrated for all of the strategies analyzed. For example, for calibration of the *Q-Strategy*, the learning rate $\beta \in [0.1, 0.3]$, the exploration rate $\epsilon \in [0.1, 0.3]$ and $\gamma \in [0.1, 0.9]$ have been varied for the applied value ranges in the literature. The selection of the *Q-Strategy* parameters constituted a trade-off between the ability to consider the dynamics of the environment and the reaction to changing conditions. Whiteson and Stone (2006) proposed an evolutionary search of Q-Learning parameters for the mountain car and server application scheduling scenarios. Sun and Peterson (1999) varied the learning rate through a heuristic policy. In Even-Dar et al. (2003) and Even-Dar and Mansour (2004) a stochastic search for estimating Q-Learning parameters in stationary settings was performed.

A detailed overview of the sensitivity analysis outcomes and resulting initialization of the parameters of the bidding strategies is presented in Appendix A. Based on the results of a sensitivity analysis, the learning rate of the *Q-Strategy* was set to $\beta = 0.1$. With an exploration rate of $\epsilon = 0.3$, the *Q-Strategy* explores the strategy space of different bids. During the experiments, the exploration rate ϵ does not change in order to preserve the learning and adaption facility, and to react to the changing market dynamics. The *Q-Strategy* discount factor $\gamma = 0.9$, which is also commonly used in the literature, assigns a higher weight to long-term gains, and accepting short-term losses. The parameters of the ZIP and GD bidding strategies have been set according to the results of the sensitivity analysis and the ranges specified in Cliff and Bruten (1997) and Gjerstad (2003).

6.1.6 Correlation Analysis

The results of the experimental settings are organized according to the criteria in Table 6.1: The type of job profile applied (LLNL or HPC2N), the number of provider

machine agents (50 or 100), as well as the provider's bidding strategy (ZIP, GD or *Q-Strategy*). In the case of consumers, the outcomes of the jobs executions are calculated with the consumer's scoring function in Table 6.1. The total outcome of the three bidding strategies can be compared directly only for homogeneous settings. In heterogeneous settings, however, the agents are assigned different proportions of *Q-Strategy* and benchmark agents. Therefore, their performances can only be directly compared with the average job score through the same types of agents since the 7 *Q-Strategy agents* execute more jobs than the 3 ZIP agents, assuming that the jobs from the workload profiles are uniformly distributed to the agents. Therefore, the *Consumers' Aggregated Average Scores* or *CAAS* is the metric applied to compare the outcomes of all consumers of the same strategy type in both homogeneous and heterogeneous consumer settings. The *Aggregation* in the CAAS points to the fact that the averages are calculated of one, two or more consumer agents of the same type.

Analogous to the consumer outcomes, the provider outcomes are similarly averaged and compared among the three bidding strategies with the *Providers' Aggregated Average Score* or *PAAS* metric according to the provider's scoring function in Table 6.1. In contrast to consumers, however, all the providers apply the same type of bidding strategy in each setting ("homogenous" providers). Therefore, the provider outcomes can be directly compared for the three bidding strategies and different settings. In contrast to CAAS, the *Aggregation* in PAAS points to the fact that the averages are calculated of all provider agents in a given setting since they are all of the same type.

The last metric of the comparative analysis is the combined consumer and provider scores, called *CCPAAS*, which measures the overall welfare for each of the settings (Table 6.1).

The setting outcomes in the correlation analysis is tested for normal distribution with the *Chi-square*, *Kolmogorow-Smirnow* and *Shapiro-Wilk* tests. The tests showed that most of the clustered outcomes are likely to be normally distributed with an $\alpha = 0.05$. With respect to the failed tests for normal distribution, the correlation analysis was performed with the *nonparametric Spearman correlation coefficient*, which does not require knowledge of the probability distribution of the data. Furthermore, the *Spearman correlation coefficient* provides a good approximation of the correlation because, unlike the *Pearson correlation coefficient*, it is not affected by higher data leaps.

A positive correlation in the case of consumers means that when the number of *Q-Strategy* agents increases and the number of benchmark agents decreases, the average

job outcome (CAAS) improves; a negative correlation means that the CAAS recedes. A positive correlation in the case of providers means that when the number of *Q-Strategy* agents increases and the number of benchmark agents falls, the average provider profit (PAAS) improves; a negative correlation means that the provider's average profit recedes. Finally, a positive correlation in the combined consumer and provider scores (CCPAAS) means that when the number of *Q-Strategy* agents increases and the number of benchmark agents decreases, the welfare in the system improves; a negative correlation means that the welfare recedes.

6.2 Evaluation Results

6.2.1 Consumer Outcomes

The overall performance of the *Q-Strategy* against the benchmark strategies in all settings is summarized in Table 6.2. The table shows the results from a consumers' perspective. The consumer outcomes are ordered according to the executed data profiles – LLNL and HPC2N, the number of providers – 50 and 100, and whether they stem from the homogeneous strategy settings or heterogeneous ones. In general, the *Q-Strategy* outperformed the benchmark strategies ZIP and GD in more than 70% of all settings.

The outcomes in the *homogeneous lower supply settings of 50 providers*, homogeneous LLNL–50 and HPC2N–50, identify a clear pattern in which *Q-Strategy* consumer agents are 100% successful against the ZIP and GD consumers in cases where providers also apply the *Q-Strategy*. However, in the same type of settings, but with ZIP providers, the GD and ZIP consumers outperformed the *Q-Strategy* consumers. In the cases with 50 GD providers, only the ZIP consumers succeeded in outperforming the *Q-Strategy* consumers; the GD consumers failed against the *Q-Strategy* consumers. This was to be expected because ZIP and GD agents are designed to adapt quickly to the public signals, bids and clearing prices of the other agents, whereas, *Q-Strategy* is designed to adapt on a long-term basis, as well as to local information and experience. Therefore, *Q-Strategy* would not be a good choice in highly competitive settings (low supply and high demand) and perfect information about other agents' actions.

In the *heterogeneous lower supply settings of 50 providers*, the *Q-Strategy* consumer agents almost outperformed the ZIP and GD consumers with the LLNL job profile, but exhibited mixed performance results in the HPC2N case. In the latter case, the *Q-Strategy* consumers failed against ZIP consumers in settings where the providers were

Table 6.2: Winning performance of the *Q-Strategy* against the benchmark strategies, ZIP and GD, in homogeneous and heterogeneous settings of consumers and providers

Data Profile		Homogeneous Settings	Heterogeneous Settings
<i>LLNL – 50</i>			
<i>Consumers</i>	<i>Providers</i>		
(<i>Q:ZIP</i>)	<i>ZIP</i>	0%	48.34%
(<i>Q:ZIP</i>)	<i>GD</i>	0%	96.67%
(<i>Q:ZIP</i>)	<i>Q</i>	100%	100%
(<i>Q:GD</i>)	<i>ZIP</i>	0%	52.67%
(<i>Q:GD</i>)	<i>GD</i>	100%	99.23%
(<i>Q:GD</i>)	<i>Q</i>	100%	87.78%
<i>LLNL – 100</i>			
<i>Consumers</i>	<i>Providers</i>		
(<i>Q:ZIP</i>)	<i>ZIP</i>	100%	100%
(<i>Q:ZIP</i>)	<i>GD</i>	100%	97.89%
(<i>Q:ZIP</i>)	<i>Q</i>	100%	100%
(<i>Q:GD</i>)	<i>ZIP</i>	100%	97.56%
(<i>Q:GD</i>)	<i>GD</i>	100%	97.56%
(<i>Q:GD</i>)	<i>Q</i>	100%	56.45%
<i>HPC2N – 50</i>			
<i>Consumers</i>	<i>Providers</i>		
(<i>Q:ZIP</i>)	<i>ZIP</i>	0%	0%
(<i>Q:ZIP</i>)	<i>GD</i>	0%	61%
(<i>Q:ZIP</i>)	<i>Q</i>	100%	99.23%
(<i>Q:GD</i>)	<i>ZIP</i>	0%	0%
(<i>Q:GD</i>)	<i>GD</i>	100%	99.45%
(<i>Q:GD</i>)	<i>Q</i>	100%	100%
<i>HPC2N – 100</i>			
<i>Consumers</i>	<i>Providers</i>		
(<i>Q:ZIP</i>)	<i>ZIP</i>	100%	100%
(<i>Q:ZIP</i>)	<i>GD</i>	100%	98.67%
(<i>Q:ZIP</i>)	<i>Q</i>	100%	100%
(<i>Q:GD</i>)	<i>ZIP</i>	0%	96.78%
(<i>Q:GD</i>)	<i>GD</i>	100%	96.34%
(<i>Q:GD</i>)	<i>Q</i>	100%	100%

also of the ZIP type. However, in the remaining settings, the *Q-Strategy* consumers showed a higher rate of success against the benchmark bidding strategies.

The consumer outcomes in the *higher supply settings of 100 providers* show that the *Q-Strategy* outperforms the benchmark bidding strategies in almost all settings and on the highest level. The *Q-Strategy* consumers failed against the GD consumers only in one setting, the one with the with the HPC2N job profile with providers of type ZIP. In summary, *Q-Strategy* is the best choice in settings with moderate competition in a scenario with 100 providers and both job profiles. In more competitive settings,

Q-Strategy is almost always the best choice in cases where providers use either the *Q-Strategy* or GD strategy. In higher competition settings with ZIP providers, the price competitive ZIP and GD strategies are the best choice for consumers.

Table 6.3 is a *Top 10* list of the combined (*Q-Strategy* and benchmark strategies) consumer outcomes for the LLNL data profile, with 50 provider agents. The highest CAAS scores were achieved in settings that are dominated by *Q-Strategy* consumer (10_Q) and provider agents of the *Q-Strategy* type. The settings [$(10_Q : 0_{GD})$ with Q-Providers] and [$(10_Q : 0_{ZIP})$ with Q-Providers] are equivalent since the the number of GD and ZIP agents in these settings is zero. The best outcome was achieved by the homogeneous settings of *Q-Strategy* agents; the subsequent highly-ranked settings are dominated by a higher number of *Q-Strategy* agents on both the consumer and provider side. Full lists of the consumer outcomes for all homogeneous and heterogenous settings can be found in Appendix B.1.

Table 6.3: Top 10 consumer outcomes of settings with 50 providers and the LLNL data profile. The higher the combined CAAS, the better the outcome of the setting.

No.	LLNL_50_Providers		Combined CAAS($\times 10^5$)
	<i>Consumers</i>	<i>Providers</i>	
1	$(10_Q:0_{GD})$	Q-Providers	-68551
	$(10_Q:0_{ZIP})$	Q-Providers	-69067
2	$(8_Q:2_{GD})$	Q-Providers	-147806
3	$(9_Q:1_{GD})$	Q-Providers	-149139
4	$(7_Q:3_{GD})$	Q-Providers	-155348
5	$(2_Q:8_{GD})$	Q-Providers	-155786
6	$(4_Q:6_{GD})$	Q-Providers	-158339
7	$(6_Q:4_{GD})$	Q-Providers	-158985
8	$(10_Q:0_{GD})$	GD-Providers	-161261
9	$(3_Q:7_{GD})$	Q-Providers	-161600
10	$(0_Q:10_{ZIP})$	ZIP-Providers	-163205

Table 6.4 shows the outcomes with the same LLNL data profile, but with a higher supply of computing services with 100 providers. Here again, the settings are dominated by the *Q-Strategy* consumers and providers that achieved the highest scores.

Table 6.5 shows the outcomes for the configuration of 50 providers and the HPC2N data profile. Similar to the LLNL data profile, in the 10 times larger HPC2N profile, the *Q-Strategy* dominated settings achieved the highest outcomes. Again, the best outcomes were achieved by the *Q-Strategy* dominated consumer and provider settings.

Table 6.4: Top 10 consumer outcomes of settings with 100 providers and the LLNL data profile. The higher the combined CAAS, the better the outcome of the setting.

No.	LLNL_100_Providers		Combined CAAS($\times 10^5$)
	<i>Consumers</i>	<i>Providers</i>	
1	(10 _Q :0 _{ZIP})	Q-Providers	-60184
	(10 _Q :0 _{GD})	Q-Providers	-60323
2	(10 _Q :0 _{GD})	ZIP-Providers	-73155
	(10 _Q :0 _{ZIP})	ZIP-Providers	-73956
3	(0 _Q :10 _{GD})	ZIP-Providers	-85604
4	(0 _Q :10 _{GD})	Q-Providers	-95449
5	(10 _Q :0 _{ZIP})	GD-Providers	-110368
	(10 _Q :0 _{GD})	GD-Providers	-110553
6	(7 _Q :3 _{GD})	Q-Providers	-124026
7	(6 _Q :4 _{GD})	Q-Providers	-129126
8	(3 _Q :7 _{GD})	Q-Providers	-136576
9	(4 _Q :6 _{GD})	Q-Providers	-141748
10	(2 _Q :8 _{GD})	ZIP-Providers	-146821

Table 6.5: Top 10 consumer outcomes of settings with 50 providers and the HPC2N data profile. The higher the combined CAAS, the better the outcome of the setting.

No.	HPC2N_50_Providers		Combined CAAS($\times 10^5$)
	<i>Consumers</i>	<i>Providers</i>	
1	(10 _Q :0 _{GD})	Q-Providers	-1054218
	(10 _Q :0 _{ZIP})	Q-Providers	-1085099
2	(0 _Q :10 _{ZIP})	GD-Providers	-1703405
3	(0 _Q :10 _{ZIP})	ZIP-Providers	-1945762
4	(10 _Q :0 _{ZIP})	GD-Providers	-1945778
5	(10 _Q :0 _{GD})	GD-Providers	-2200169
6	(9 _Q :1 _{ZIP})	Q-Providers	-2535239
7	(0 _Q :10 _{ZIP})	Q-Providers	-2632936
8	(8 _Q :2 _{ZIP})	Q-Providers	-2725949
9	(7 _Q :3 _{ZIP})	Q-Providers	-2855771
10	(1 _Q :9 _{ZIP})	GD-Providers	-3120306

Table 6.6: Top 10 consumer outcomes of settings with 100 providers and the HPC2N data profile. The higher the combined CAAS, the better the outcome of the setting.

No.	HPC2N_100_Providers		Combined CAAS($\times 10^5$)
	<i>Consumers</i>	<i>Providers</i>	
1	(10 _Q :0 _{GD})	Q-Providers	-576447
	(10 _Q :0 _{ZIP})	Q-Providers	-579618
2	(0 _Q :10 _{GD})	ZIP-Providers	-734462
3	(10 _Q :0 _{GD})	GD-Providers	-903385
4	(0 _Q :10 _{GD})	Q-Providers	-974690
5	(10 _Q :0 _{ZIP})	GD-Providers	-1028515
6	(10 _Q :0 _{ZIP})	ZIP-Providers	-1094188
7	(10 _Q :0 _{GD})	ZIP-Providers	-1107360
8	(0 _Q :10 _{GD})	GD-Providers	-1231504
9	(0 _Q :10 _{ZIP})	ZIP-Providers	-1392967
10	(1 _Q :9 _{GD})	ZIP-Providers	-1471657

The outcomes in Table 6.6 with the higher supply and HPC2N data profile confirm the effectiveness of *Q-Strategy* in the LLNL settings. In the 50 and 100 provider cases with the HPC2N data profile, the settings dominated by the *Q-Strategy* agents scored higher than those of the benchmark dominated settings.

Figure 6.3 shows the consumer outcomes in settings where *all the providers' agents are of the GD type* and the target job profile is LLNL. The sub-figures (a)–(b) display the results in higher competitive settings with 50 providers, sub-figures (c)–(d) present results in settings with 100 providers. The x-axis displays the ten different competing scenarios of 0-*Q-Strategy* and 10 *Benchmark* (ZIP or GD) consumers to 10-*Q-Strategy* and 0-*Benchmark*-consumers in all consecutive combinations. The y-axis displays consumers' aggregated average job score for each of the competing scenarios. In contrast to the previously presented “Top 10” tables these figures directly compare the averaged outcomes of the *Q-Strategy* agents to those of the *Benchmark* agents in each of the competing scenarios. For example, in sub-figure (a), the average job score of the 1-*Q-Strategy* consumers is higher than the average scores of all 9-GD agents. Moreover, in the higher competition setting of sub-figure (a) the *Q-Strategy* consumers outperform (CAAS) the GDs in the homogeneous (in these cases, a direct comparison of the 0:10 and 10:0 is performed) and heterogeneous settings (between 0:10 and 10:0). The *Spearman correlation coefficient* r_s for the *Q-Strategy* consumers is somewhat positive, which means that when the number of *Q-Strategy* agents increases, the CAAS score improves. From the perspective of the

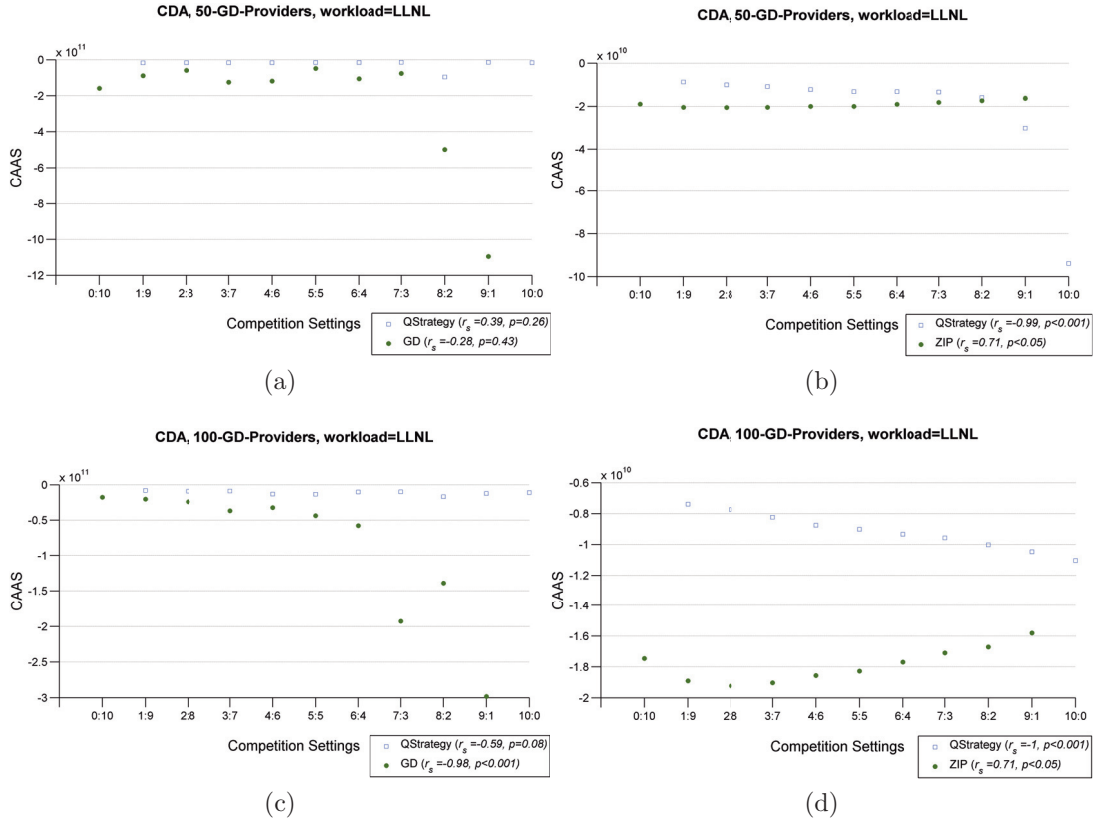


Figure 6.3: Consumer outcomes in settings with GD-Providers and the LLNL data profile: (a) Competing settings of (Q-Strategy:GD)-Consumers and 50-GD-Providers; (b) Competing settings of (Q-Strategy:ZIP)-Consumers and 50-GD-Providers; (c) Competing settings of (Q-Strategy:GD)-Consumers and 100-GD-Providers; and, (d) Competing settings of (Q-Strategy:ZIP)-Consumers and 100-GD-Providers.

GD consumers, the correlation is somewhat negative, when the number of GD agents decreases. Therefore, the CAAS score of the lesser becoming GD consumers decreases somewhat as the number of *Q-Strategy* consumers increases.

In contrast, in sub-figure (b) opponents of the *Q-Strategy* consumer agents are ZIP consumers. Here, the *Q-Strategy* consumer agents outperform the ZIP opponents in almost all heterogeneous settings, but failed in the homogeneous settings. In this setting and in settings against ZIP consumers, *Q-Strategy* consumers experienced an almost negative correlation between their CAAS scores and an increase in the number of consumers for the *Q-Strategy*. On the other hand, ZIP agents experienced

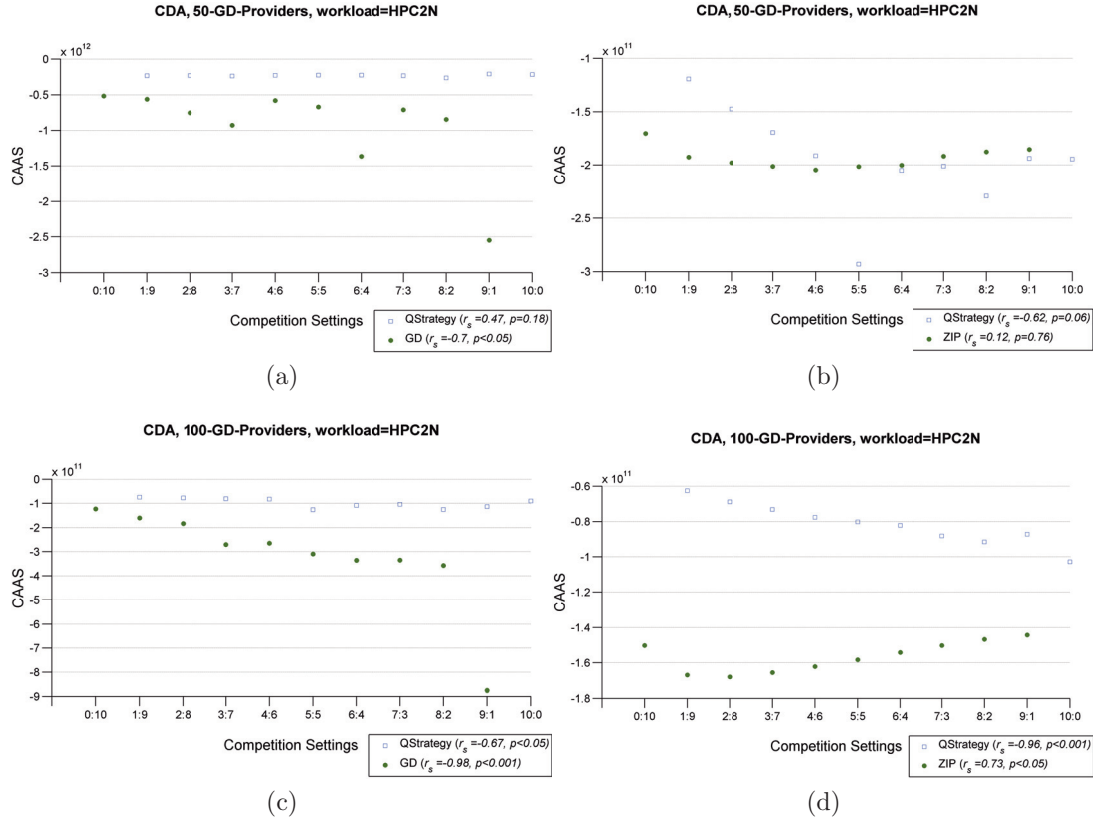


Figure 6.4: Consumer outcomes in settings with GD-Providers and the HPC2N data profile: (a) Competing settings of (Q-Strategy:GD)-Consumers and 50-GD-Providers; (b) Competing settings of (Q-Strategy:ZIP)-Consumers and 50-GD-Providers; (c) Competing settings of (Q-Strategy:GD)-Consumers and 100-GD-Providers; and, (d) Competing settings of (Q-Strategy:ZIP)-Consumers and 100-GD-Providers.

an almost positive correlation between the decreasing number of agents and CAAS scores, e.g. the 1-ZIP consumer managed to achieve a higher CAAS score than the 9-*Q-Strategy* consumers. Both correlations are significant with $p < 0.001$ and $p < 0.05$.

In the higher supply case with 100 providers, sub-figures (c)–(d), the *Q-Strategy* agents outperform the benchmark strategies ZIP and GD in all heterogeneous and homogeneous settings. In the higher supply setting, the CAAS correlation of the GD consumers changed from somewhat negative to almost negative; in other words, GD consumers achieve higher CAAS when their number increases in relation to the num-

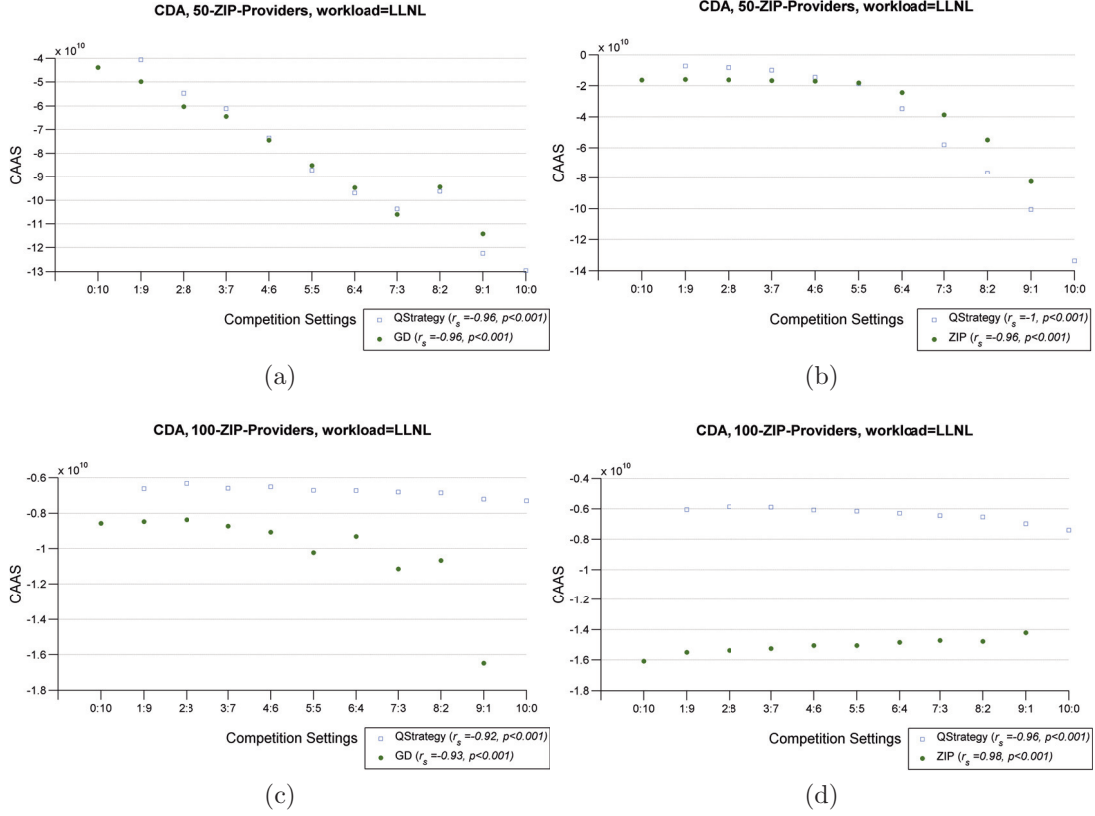


Figure 6.5: Consumer outcomes in settings with Q -Providers and the LLNL data profile: (a) Competing settings of (Q -Strategy:GD)-Consumers and 50-ZIP-Providers; (b) Competing settings of (Q -Strategy:ZIP)-Consumers and 50-ZIP-Providers; (c) Competing settings of (Q -Strategy:GD)-Consumers and 100-ZIP-Providers; and, (d) Competing settings of (Q -Strategy:ZIP)-Consumers and 100-ZIP-Providers.

ber of Q -Strategy consumers. Similar to the outcome in sub-figure (b), the increase in the number of Q -Strategy consumers correlated negatively with their CAAS score, whereas the decrease in the number of ZIP consumers improved their CAAS score.

Figure 6.4 shows the same setting configurations, but with the HPC2N job profile, which is 10 times larger. In almost all heterogeneous and homogeneous settings, the Q -Strategy agents achieved a higher CAAS than those with the benchmark strategies, except for the setting in sub-figure (b), where the ZIP consumers scored higher, on average, in settings with a higher number of Q -Strategy agents. As discussed before,

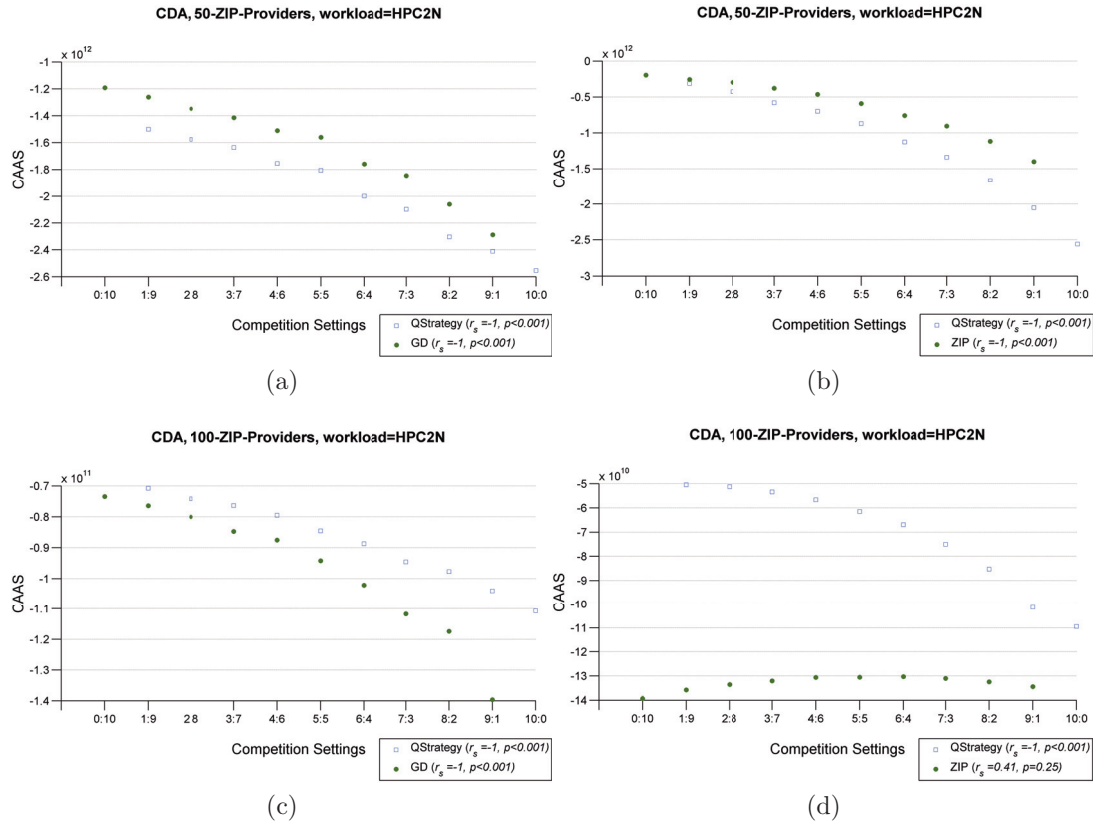


Figure 6.6: Consumer outcomes in settings with ZIP-Providers and the HPC2N data profile: (a) Competing settings of (Q-Strategy:GD)-Consumers and 50-ZIP-Providers; (b) Competing settings of (Q-Strategy:ZIP)-Consumers and 50-ZIP-Providers; (c) Competing settings of (Q-Strategy:GD)-Consumers and 100-ZIP-Providers; and, (d) Competing settings of (Q-Strategy:ZIP)-Consumers and 100-ZIP-Providers.

the ZIP strategy adapts quickly in highly competitive settings, which results in higher CAAS scores. Furthermore, the outcomes in sub-figures (b)–(d) were, on average, almost negatively correlated for the *Q-Strategy*. The correlation of GD consumers with a higher number of *Q-Strategy* agents is almost negative, but positive, vice versa. In contrast, an almost positive correlation can be observed in ZIP consumer agents as the number of *Q-Strategy* agents increases.

The competitive property of the ZIP and GD strategies becomes evident when the

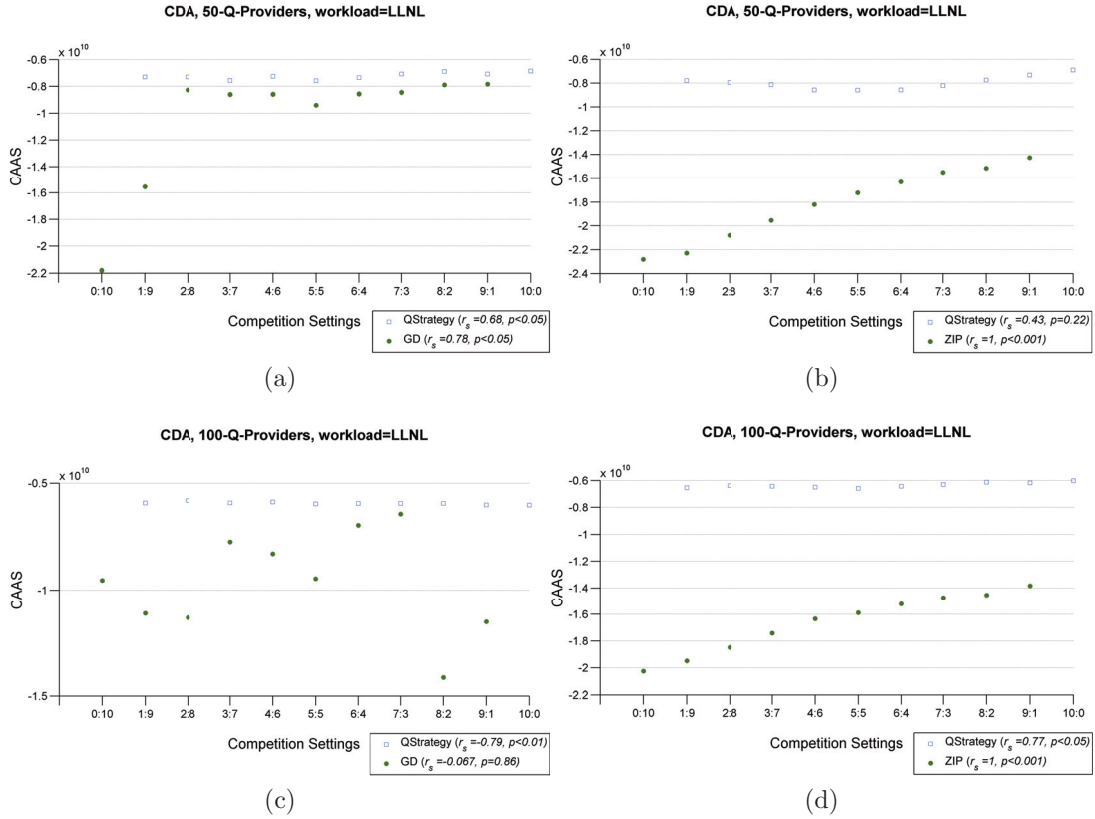


Figure 6.7: Consumer outcomes in settings with Q-Providers and the LLNL data profile: (a) Competing settings of (Q-Strategy:GD)-Consumers and 50-Q-Providers; (b) Competing settings of (Q-Strategy:ZIP)-Consumers and 50-Q-Providers; (c) Competing settings of (Q-Strategy:GD)-Consumers and 100-Q-Providers; and, (d) Competing settings of (Q-Strategy:ZIP)-Consumers and 100-Q-Providers.

providers are also type ZIP (Figure 6.5). In the smaller supply settings with 50 providers, the *Q-Strategy* consumers outperform their benchmark opponents only in the heterogeneous settings with a smaller number of *Q-Strategy* agents; in settings with a higher number of *Q-Strategy* agents, the competitive benchmark agents ZIP and GD achieved higher CAAS. However, in the higher supply settings with 100 providers, the *Q-Strategy* agents outperform the benchmarks in all homogeneous and heterogeneous settings. In the case of ZIP providers, the increase in the number of *Q-Strategy* agents almost had a negative impact on their CAAS outcome.

In the larger data profile in Figure 6.6 a lower supply of ZIP providers (high com-

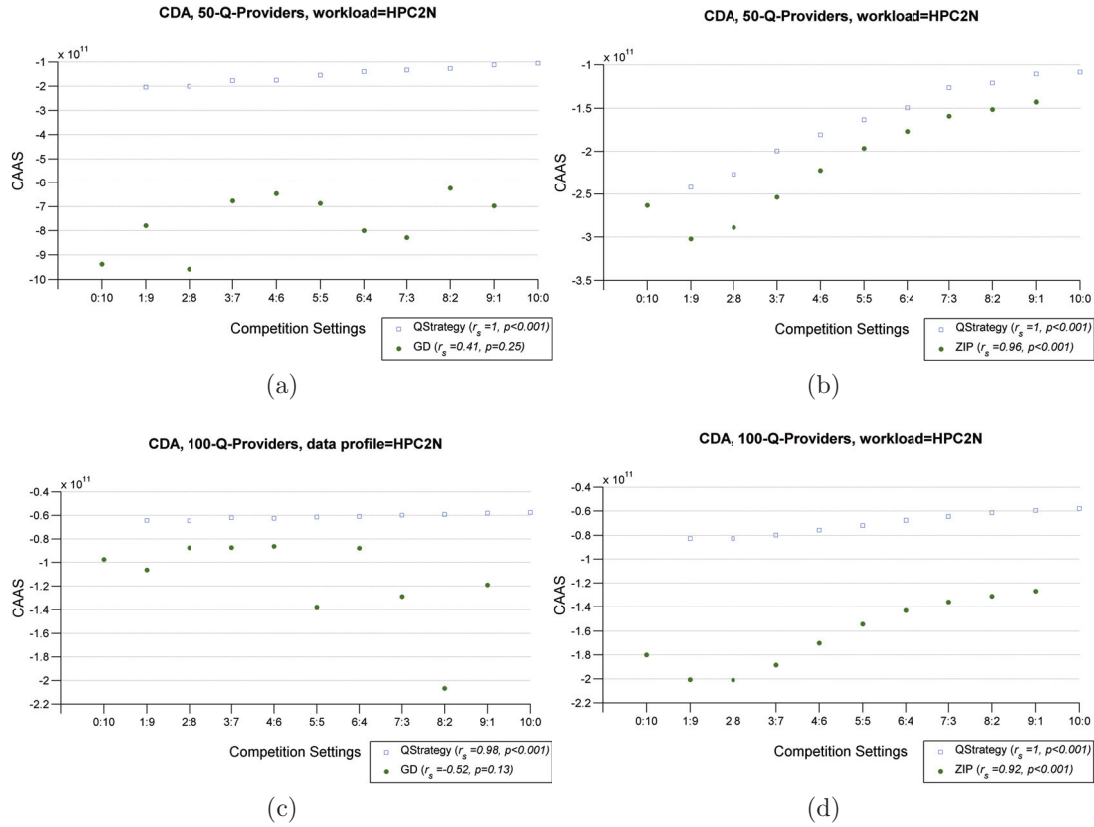


Figure 6.8: Consumer outcomes in settings with Q -Providers and the HPC2N data profile: (a) Competing settings of (Q -Strategy:GD)-Consumers and 50- Q -Providers; (b) Competing settings of (Q -Strategy:ZIP)-Consumers and 50- Q -Providers; (c) Competing settings of (Q -Strategy:GD)-Consumers and 100- Q -Providers; and, (d) Competing settings of (Q -Strategy:ZIP)-Consumers and 100- Q -Providers.

petition for less computing services), the GD and ZIP benchmark strategies outperformed the Q -Strategy, whereas in the higher supply setting, the Q -Strategy consumers performed better than the benchmark strategies. However, the rising number of Q -Strategy agents is almost negatively correlated for the overall consumer scores (CAAS); only the ZIP consumers in (d) achieved a somewhat positive correlation with an increase in the number of Q -Strategy agents.

Figure 6.7 depicts a case, in which the providers are of the Q -Strategy type. As stated earlier, the fully homogeneous settings of Q -Strategy consumers and Q -Strategy

providers achieved the highest scores. Here, this result is confirmed; in sub-figures (a)–(d) the *Q-Strategy* consumers achieved a higher CAAS score against the benchmark bidding strategies in the low and high supply settings. In contrast to the settings of ZIP and GD providers, here the consumer bidding strategies correlated positively with an increase in the number of *Q-Strategy* agents. The GD consumers in the high supply setting do not yield significant evidence ($p = 0.86$) for a correlation ($r_s = -0.06$ is very close to a correlation of 0).

Figure 6.8 also shows the positive results for the larger HPC2N data profile. In the case of *Q-Strategy* providers, the *Q-Strategy* consumers outperformed the benchmark strategies GD and ZIP. Furthermore, the correlation is almost positive for the consumers' CAAS when the number of *Q-Strategy* agents increases, whereas the correlation for ZIP consumers is almost positive when the *Q-Strategy* consumers are in the majority.

6.2.2 Provider Outcomes

Table 6.7 presents the outcomes from the provider's perspective with a supply of 50 machines and with the LLNL data profile. All providers are classic profit maximizers and *Q-Strategy* is configured with the profit maximizing scoring function for the provider case. As shown in the "Top 10" outcomes, Table 6.7, the highest providers' aggregated average score (PAAS) was achieved by providers applying the ZIP strategy. However, the higher PAAS scores are achieved in settings dominated by *Q-Strategy* consumers. Therefore, *Q-Strategy* consumers paid higher prices, on average, which resulted in higher profits for the providers. This can be explained by the fact that *Q-Strategy* consumers optimize their job specific goals in terms of completion time and payments, which resulted in higher payments than those of the price competitive ZIP and GD consumers. Moreover, reinforcement-based strategies optimize long-term rewards rather than short-term gains like in cases using the ZIP and GD strategies (Note: GD and ZIP assume and use the public information of all other agents' bids and clearing prices in order to remain price competitive. In markets with imperfect information, these strategies might not be the best choice). Full lists of the providers' outcomes of all homogeneous and heterogeneous settings can be found in Appendix B.2.

In the case with 100 providers and the LLNL data profile, Table 6.8, the highest profit was achieved by providers applying the GD strategy, but again in settings dominated by *Q-Strategy* consumer agents.

Table 6.7: Top 10 provider outcomes of settings with 50 providers and the LLNL data profile. The higher the PAAS, the better the outcome of the setting.

No.	LLNL_50_Providers		PAAS($\times 10^5$)
	<i>Consumers</i>	<i>Providers</i>	
1	(10 _Q :0 _{ZIP})	ZIP-Providers	6180
	(10 _Q :0 _{GD})	ZIP-Providers	6135
2	(9 _Q :1 _{GD})	ZIP-Providers	6088
3	(9 _Q :1 _{ZIP})	ZIP-Providers	6018
4	(7 _Q :3 _{GD})	ZIP-Providers	5977
5	(8 _Q :2 _{GD})	ZIP-Providers	5937
6	(6 _Q :4 _{GD})	ZIP-Providers	5867
7	(5 _Q :5 _{GD})	ZIP-Providers	5841
8	(8 _Q :2 _{ZIP})	ZIP-Providers	5779
9	(4 _Q :6 _{GD})	ZIP-Providers	5712
10	(7 _Q :3 _{ZIP})	ZIP-Providers	5595

Table 6.8: Top 10 provider outcomes of settings with 100 providers and the LLNL data profile. The higher the PAAS, the better the outcome of the setting.

No.	LLNL_100_Providers		PAAS($\times 10^5$)
	<i>Consumers</i>	<i>Providers</i>	
1	(10 _Q :0 _{GD})	GD-Providers	2095
	(10 _Q :0 _{ZIP})	GD-Providers	2079
2	(9 _Q :1 _{ZIP})	GD-Providers	2029
3	(9 _Q :1 _{GD})	GD-Providers	2023
4	(8 _Q :2 _{GD})	GD-Providers	1953
5	(8 _Q :2 _{ZIP})	GD-Providers	1934
6	(7 _Q :3 _{GD})	GD-Providers	1887
7	(7 _Q :3 _{ZIP})	GD-Providers	1833
8	(6 _Q :4 _{GD})	GD-Providers	1804
9	(5 _Q :5 _{GD})	GD-Providers	1734
10	(6 _Q :4 _{ZIP})	GD-Providers	1732

Table 6.9: Top 10 provider outcomes of settings with 50 providers and the HPC2N data profile. The higher the PAAS, the better the outcome of the setting.

No.	HPC2N_50_Providers		PAAS($\times 10^5$)
	<i>Consumers</i>	<i>Providers</i>	
1	(10 _Q :0 _{GD})	ZIP-Providers	53314
	(10 _Q :0 _{ZIP})	ZIP-Providers	53066
2	(9 _Q :1 _{GD})	ZIP-Providers	52757
3	(8 _Q :2 _{GD})	ZIP-Providers	52172
4	(9 _Q :1 _{ZIP})	ZIP-Providers	51785
5	(7 _Q :3 _{GD})	ZIP-Providers	51670
6	(6 _Q :4 _{GD})	ZIP-Providers	50804
7	(8 _Q :2 _{ZIP})	ZIP-Providers	50281
8	(5 _Q :5 _{GD})	ZIP-Providers	49697
9	(4 _Q :6 _{GD})	ZIP-Providers	49222
10	(7 _Q :3 _{ZIP})	ZIP-Providers	48436

Table 6.10: Top 10 provider outcomes of settings with 100 providers and the HPC2N data profile. The higher the PAAS, the better the outcome of the setting.

No.	HPC2N_100_Providers		PAAS($\times 10^5$)
	<i>Consumers</i>	<i>Providers</i>	
1	(10 _Q :0 _{GD})	GD-Providers	17669
	(10 _Q :0 _{ZIP})	GD-Providers	17619
2	(9 _Q :1 _{GD})	GD-Providers	17122
3	(9 _Q :1 _{ZIP})	GD-Providers	17049
4	(8 _Q :2 _{GD})	GD-Providers	16593
5	(8 _Q :2 _{ZIP})	GD-Providers	16408
6	(7 _Q :3 _{GD})	GD-Providers	16139
7	(7 _Q :3 _{ZIP})	GD-Providers	15631
8	(6 _Q :4 _{GD})	GD-Providers	15459
9	(5 _Q :5 _{GD})	GD-Providers	14843
10	(6 _Q :4 _{ZIP})	GD-Providers	14761

Tables 6.9 and 6.10 display the results with 50 and 100 providers with the HPC2N data profile. The outcomes of the LLNL and HPC2N data profiles are comparable, but resulted in higher profits due to higher (10 times more) number of jobs. In the 50 provider HPC2N settings the ZIP providers again achieved higher profits than the GD and *Q-Strategy* providers (see Appendix B for a full version of the table), whereas the GD providers outperformed the GD and *Q-Strategy* providers in the 100 provider HPC2N settings. Like in the 50 provider settings, the profitable provider settings also consisted of a dominant number of *Q-Strategy* consumers.

Figure 6.9 presents the provider outcomes for the LLNL data profile and the varying types of consumer agents. Based on the design specification and focus on consumers, all providers are modeled to be homogeneous. Therefore, their outcomes with ZIP, GD and *Q-Strategy* provider agents can be directly compared. Unlike with consumers, in the case with provider agents, the scoring function of *Q-Strategy* is to maximize profit.

The first sub-figure (a) shows a setting with ($Q : GD$)-consumers, 50 providers in total for all three strategy types – ZIP, GD and *Q-Strategy* – and the data profile LLNL. The x-axis represents the ($Q : GD$)-consumers competition settings and the y-axis is the providers' aggregated average score. Here, the price competitive and profit maximizing bidding strategies, ZIP and GD, outperformed *Q-Strategy* in all consumer competition settings. However, all three strategies in all of the four sub-figures correlate significantly and almost positively with an increasing number of *Q-Strategy* consumer agents. In the case of sub-figure (b), the *Q-Strategy* provider agents outperformed the ZIP and GD providers in the setting, fully dominated by ZIP consumers outperforming ZIP providers in the ($1_Q : 9_{ZIP}$)-consumers setting. With an increasing number of *Q-Strategy* consumers, the ZIP providers outperformed the GD providers in the higher competition settings with 50 providers, but lost against them in the moderate competition settings with 100 providers.

In the case of settings with a higher supply, sub-figures (c)–(d), the *Q-Strategy* provider agents lost in most of the competitions against the benchmark agents, but have been successful against the ZIP providers (also in a case against a GD provider, sub-figure (d)) in settings with a smaller number of *Q-Strategy* consumers.

In the case of Figure 6.10 and the larger data profile HPC2N, the results are similar. However, in sub-figure (d) the *Q-Strategy* providers achieved a higher PAAS against the ZIP providers in most of the competition settings. Similar to the previous outcome with the LLNL data profile, the *Q-Strategy* providers were successful in this

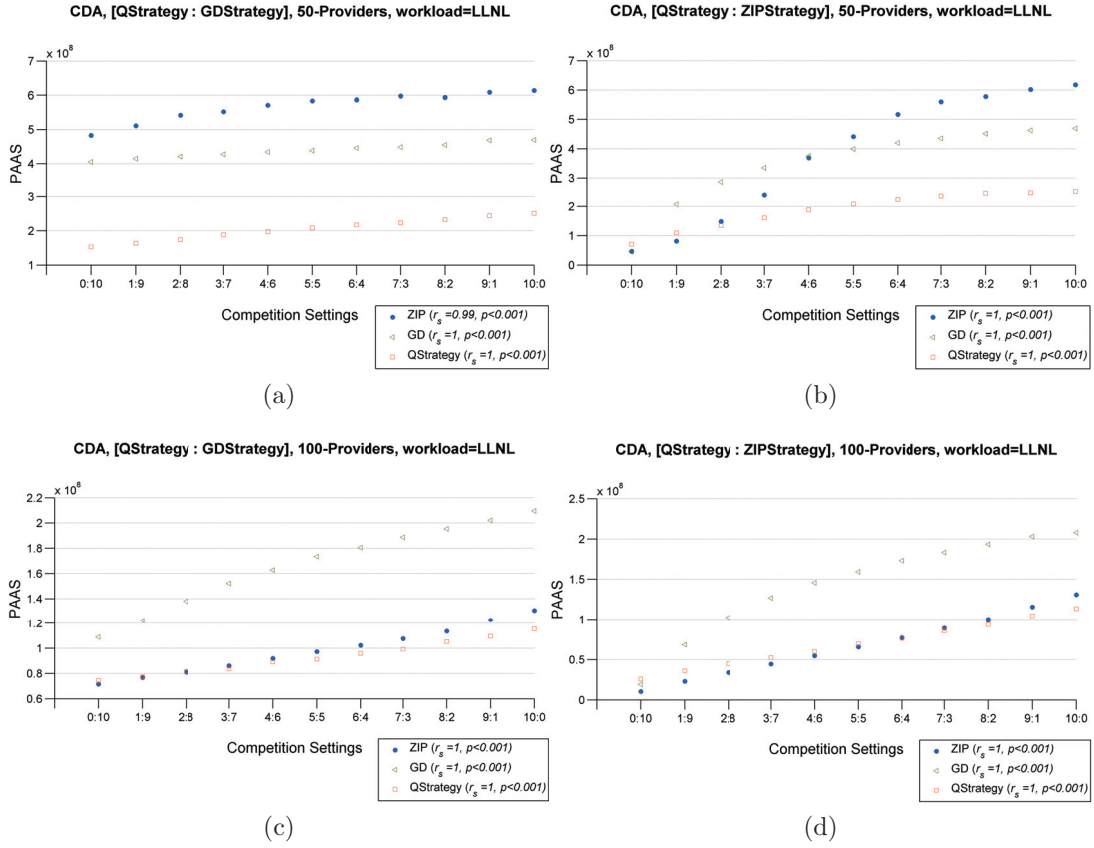


Figure 6.9: Provider outcomes in settings with competing consumers and the LLNL data profile: (a) Outcome comparison between the 50-Providers of types ZIP, GD and *Q-Strategy*, and competing consumers of type (Q-Strategy:GD); (b) Outcome comparison between the 50-Providers of types ZIP, GD and *Q-Strategy*, and competing consumers of type (Q-Strategy:ZIP); (c) Outcome comparison between the 100-Providers of types ZIP, GD and *Q-Strategy*, and competing consumers of type (Q-Strategy:GD); and, (d) Outcome comparison between the 100-Providers of types ZIP, GD and *Q-Strategy*, and competing consumers of type (Q-Strategy:ZIP).

setting, which was dominated by ZIP consumers. However, this evidence is observable only in the moderate competition settings of 100 providers; in the case of higher competition (lower supply of 50 providers), the *Q-Strategy* providers have been successful only in settings, which are fully dominated by ZIP consumers. As soon as the number of *Q-Strategy* consumers increases in relation to the ZIP consumers, the *Q-*

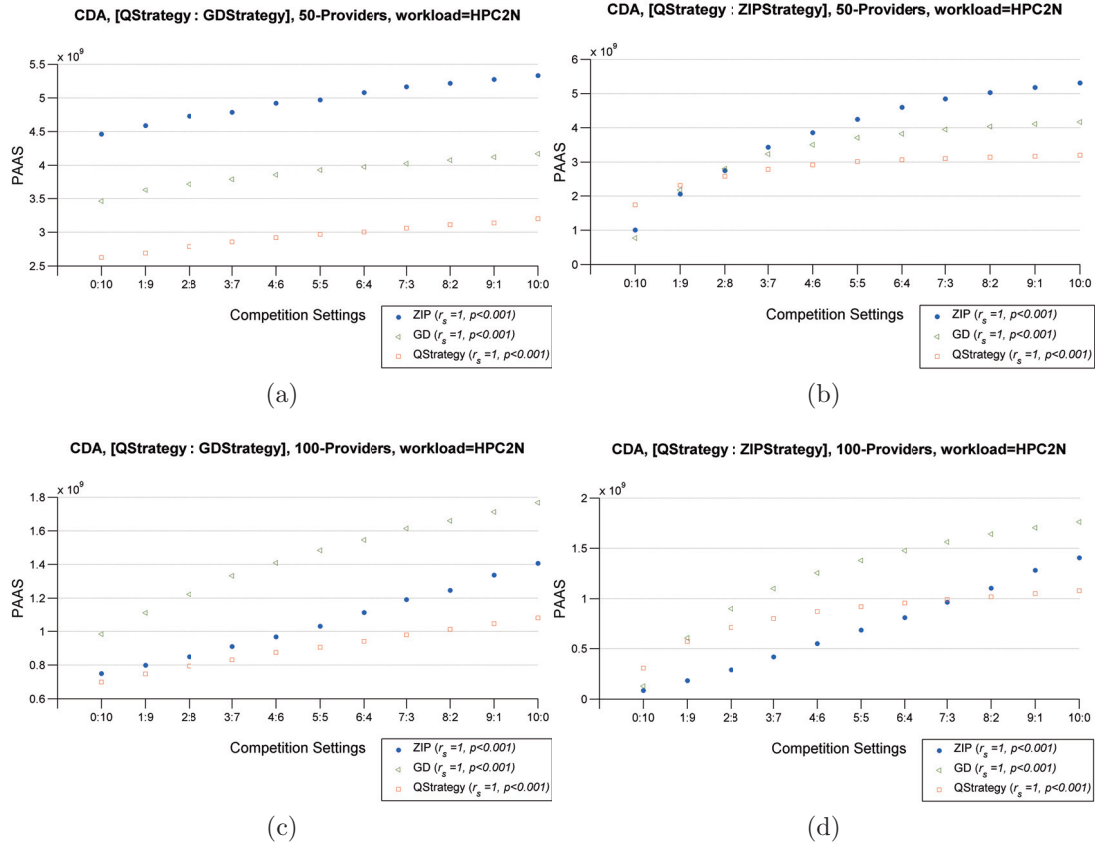


Figure 6.10: Provider outcomes in settings with competing consumers and the HPC2N data profile: (a) Outcome comparison between the 50-Providers of types ZIP, GD and *Q-Strategy*, and competing consumers of type (Q-Strategy:GD); (b) Outcome comparison between the 50-Providers of types ZIP, GD and *Q-Strategy*, and competing consumers of type (Q-Strategy:ZIP); (c) Outcome comparison between the 100-Providers of types ZIP, GD and *Q-Strategy*, and competing consumers of type (Q-Strategy:GD); and, (d) Outcome comparison between the 100-Providers of types ZIP, GD and *Q-Strategy*, and competing consumers of type (Q-Strategy:ZIP).

Strategy providers tend to be outperformed by the ZIP and GD providers. Moreover, in all settings the rising number of learning *Q-Strategy* consumers strongly correlated with the PAAS scores of all provider types (ZIP, GD and *Q-Strategy*). Therefore, in all cases, the providers will prefer to have consumers of type *Q-Strategy*, which maximize their goals in the long term.

The outcomes of this section confirm and enhance the findings of Gjerstad (2003), Tesauro and Bredin (2002), and Das et al. (2001) that the ZIP and GD strategies are good candidates in price competitive markets, in which public information of other agents' actions is available. In addition, these analyses shows that the ZIP providers outperform the GD providers in the highly competitive settings with 50 providers and in both workload profiles, LLNL and HPC2N. However, in the moderately competitive settings with 100 providers, the GD providers received the highest PAAS scores on average. Moreover, an important result of this section is that all types of the elaborated provider agents (ZIP, GD, *Q-Strategy*) benefit with the an increasing number of Q-Strategy consumers that aim to maximize their complex long term goals ("minimum completion time and payments"). In settings, dominated by profit maximizing ZIP and GD agents, agents, however, the PAAS score for providers was significantly ($p < 0.001$) lower.

6.2.3 Combined Outcomes

The combination of the consumers' and providers' aggregated average scores (CC-PAAS) provides a general view of the total CAAS score of both the *Q-Strategy* CAAS and Benchmark CAAS, as well as the total PAAS of the providers. Since the consumers' CAAS has a higher value than the PAAS, with an exponential factor difference of at least 10^2 , the "Top 10 Lists" would look similar to those of the consumer outcomes. However, the combined analysis shows the total outcome for each setting, which is summarized for all consumer and provider agents into one value, the CCPAAS score. Therefore, the CCPAAS analysis allows the derivation of evidence regarding the general effects of the *Q-Strategy* agents when competing against benchmark agents in homogeneous and heterogeneous scenarios.

Figure 6.11 depicts the combined outcomes in the settings with competing [Q:GD] and [Q:ZIP] consumers, 50 and 100 providers of type GD, and with the LLNL workload profile. The combined outcomes in sub-figure (a) do not significantly ($p = 0.92$) correlate with an increasing number of *Q-Strategy* consumers against a decreasing number of GD consumers and in the more competitive setting with 50 providers. In the same setting, but with ZIP *Benchmark* consumers (sub-figure (b)), the CCPAAS correlate significantly and almost negatively with an increasing number of *Q-Strategy* consumers. In the moderately competitive setting with 100 providers, sub-figures (c) and (d), the correlation relationships reverse: sub-figure (c) shows a negative correlation with the increasing number of *Q-Strategy* consumers in relation to the GD consumers and the CCPAAS score, whereas sub-figure (d) does not show any sig-

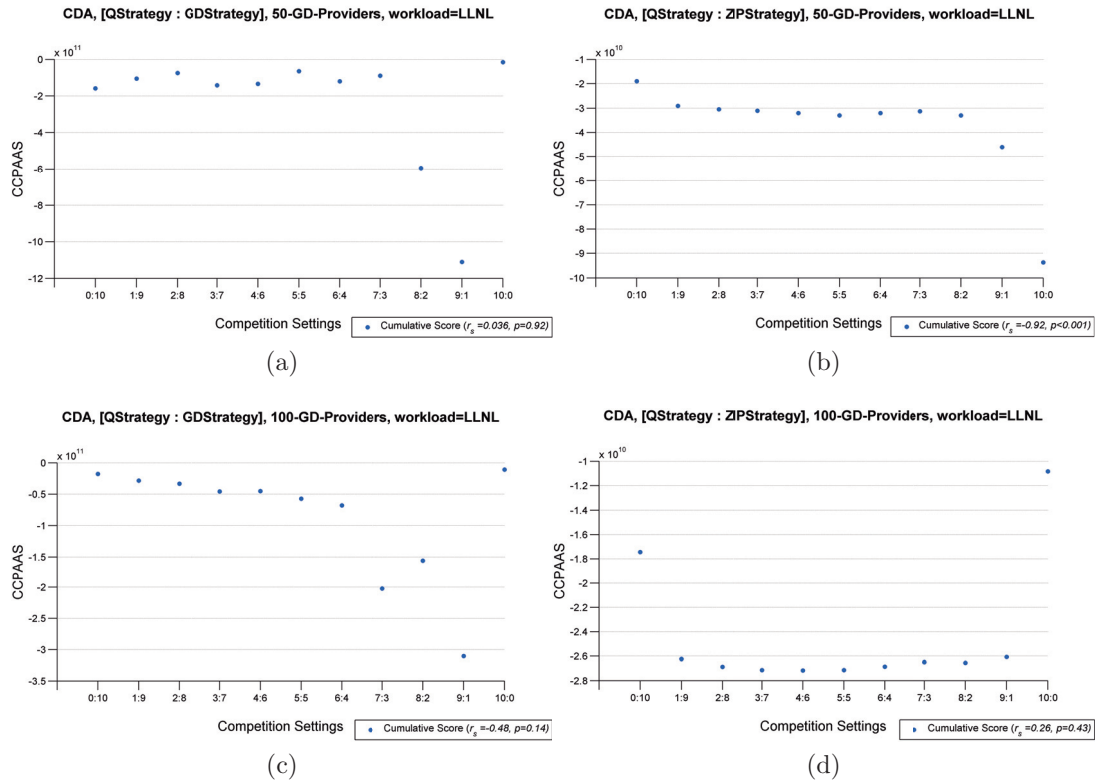


Figure 6.11: Combined consumer and provider outcomes in settings with GD-Providers and the LLNL data profile: (a) Competing settings of (Q-Strategy:GD)-Consumers and 50-GD-Providers; (b) Competing settings of (Q-Strategy:ZIP)-Consumers and 50-GD-Providers; (c) Competing settings of (Q-Strategy:GD)-Consumers and 100-GD-Providers; and, (d) Competing settings of (Q-Strategy:ZIP)-Consumers and 100-GD-Providers.

nificant change of direction of the CCPAAS scores with an increasing number of *Q-Strategy* consumers in relation to ZIP consumers.

In similar settings, but with the larger HPC2N data profile, an increasing number of *Q-Strategy* agents does not correlate with the CCPAAS (Figure 6.12). Moreover, the sub-figures (a) to (d) do not show any significant correlation of the CCPAAS with an increasing number of *Q-Strategy* agents, however, in the more competitive settings with 50 providers, the CCPAAS tend to correlate negatively.

Figure 6.13 shows the similar cases with the LLNL workload profile and with type ZIP

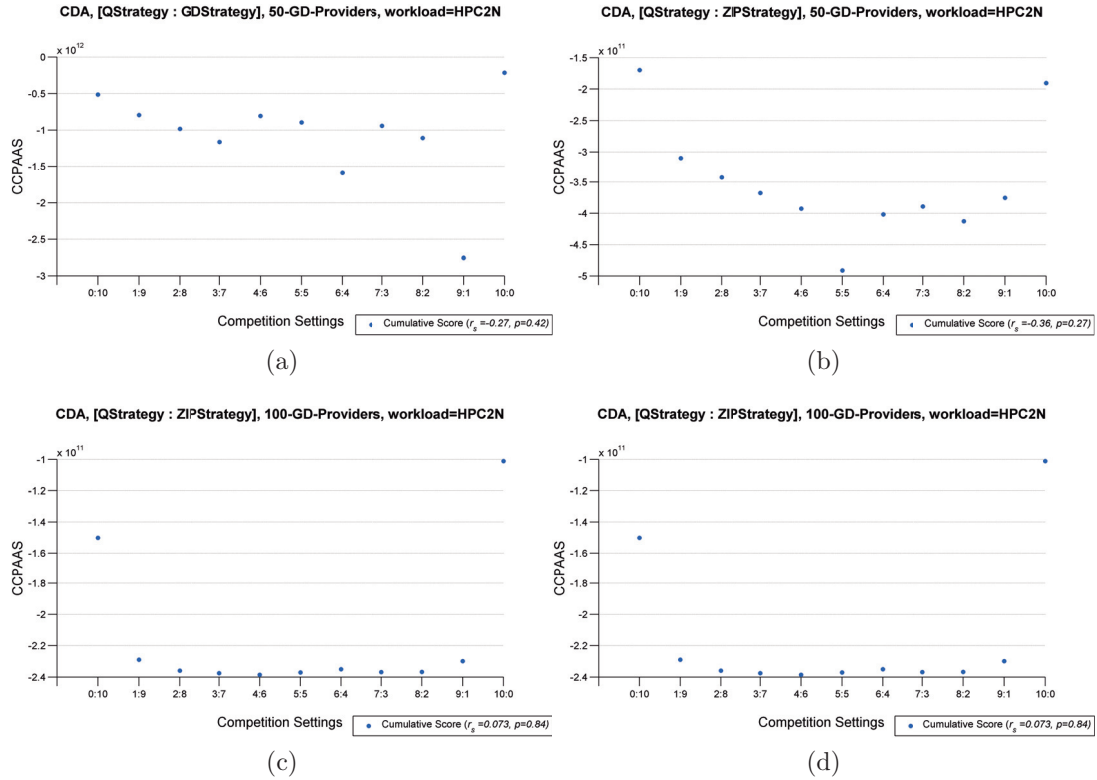


Figure 6.12: Combined consumer and provider outcomes in settings with GD-Providers and the HPC2N data profile: (a) Competing settings of (Q-Strategy:GD)-Consumers and 50-GD-Providers; (b) Competing settings of (Q-Strategy:ZIP)-Consumers and 50-GD-Providers; (c) Competing settings of (Q-Strategy:GD)-Consumers and 100-GD-Providers; and, (d) Competing settings of (Q-Strategy:ZIP)-Consumers and 100-GD-Providers.

providers. Sub-figures (a) and (b) show that the CCPAAS correlate significantly and negatively with an increasing number of *Q-Strategy* consumers. As observed in the consumer outcomes, the ZIP and GD consumers outperformed the *Q-Strategy* when all providers applied the ZIP strategy. Therefore, sub-figures (a) and (b) confirm this relationship since the CCPAAS scores improved when the number of ZIP consumers increase in relation to the *Q-Strategy* consumers in settings with ZIP providers. In the moderately competitive settings with 100 providers, sub-figures (c) and (d), there is no observable or significant correlation between the CCPAAS and the increasing number of *Q-Strategy* consumers.

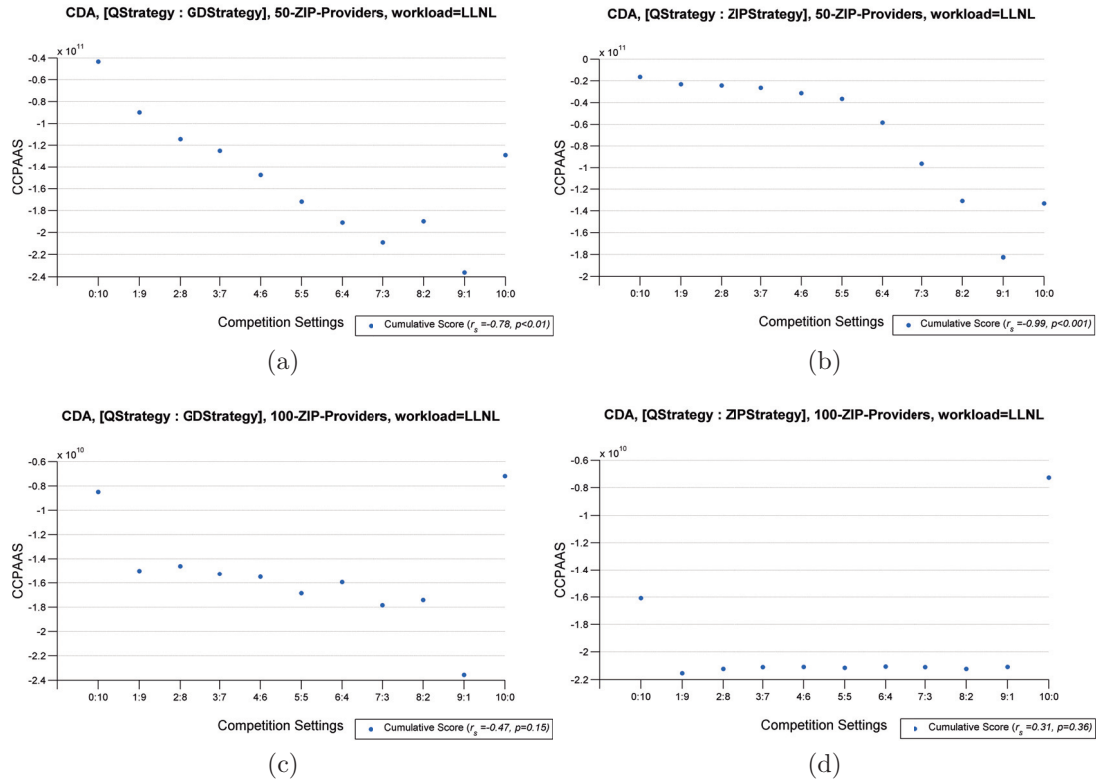


Figure 6.13: Combined consumer and provider outcomes in settings with ZIP-Providers and the LLNL data profile: (a) Competing settings of (Q-Strategy:GD)-Consumers and 50-ZIP-Providers; (b) Competing settings of (Q-Strategy:ZIP)-Consumers and 50-ZIP-Providers; (c) Competing settings of (Q-Strategy:GD)-Consumers and 100-ZIP-Providers; and, (d) Competing settings of (Q-Strategy:ZIP)-Consumers and 100-ZIP-Providers.

Figure 6.14 shows the same settings, but with the HPC2N job profile. Only sub-figure (b) shows a significant and negative correlation of the CCPAAS and the increasing number of *Q-Strategy* consumers in relation to ZIP consumers. The sub-figures (a), (c) and (d) show a tendency to negative correlations of the CCPAAS and the competing consumer settings since the correlations are not significant ($p > 0.05$).

Figure 6.15 shows the case, in which all providers apply the *Q-Strategy* and the consumers are executed with the LLNL workload. The outcomes in the high competitive settings, sub-figures (a) and (b), show a significant ($p < 0.05$) and positive correlation

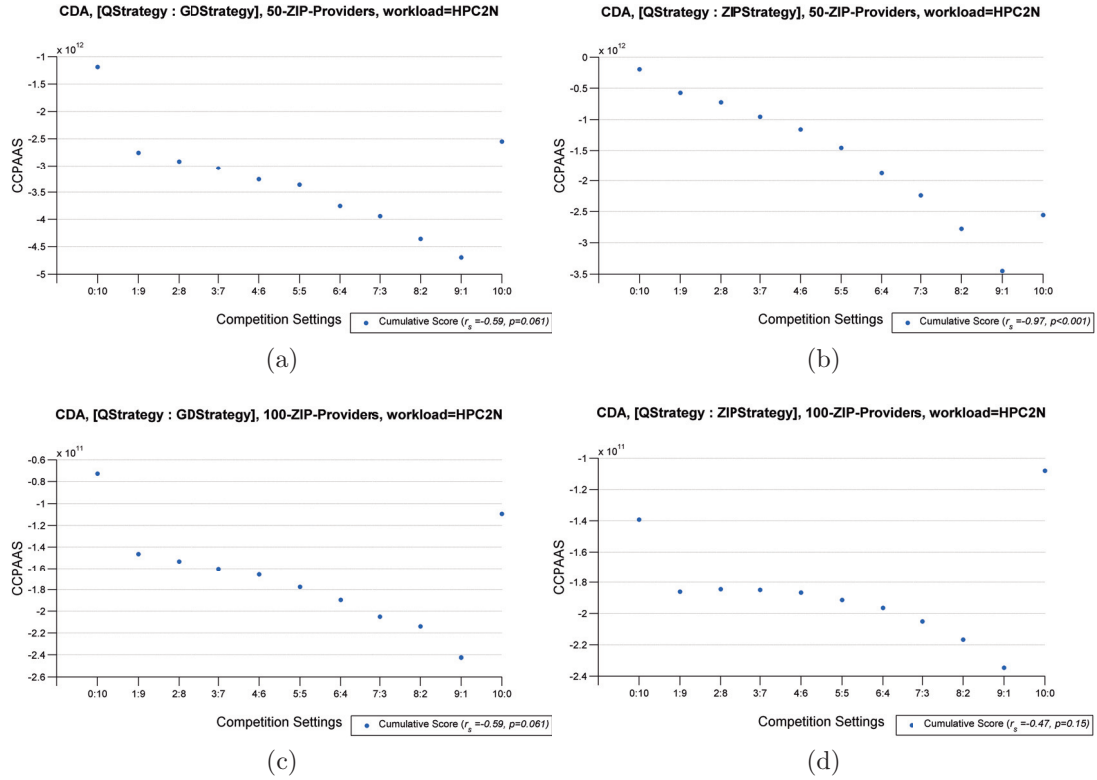


Figure 6.14: Combined consumer and provider outcomes in settings with ZIP-Providers and the HPC2N data profile: (a) Competing settings of (Q-Strategy:GD)-Consumers and 50-ZIP-Providers; (b) Competing settings of (Q-Strategy:ZIP)-Consumers and 50-ZIP-Providers; (c) Competing settings of (Q-Strategy:GD)-Consumers and 100-ZIP-Providers; and, (d) Competing settings of (Q-Strategy:ZIP)-Consumers and 100-ZIP-Providers.

between the CCPAAS and increasing number of *Q-Strategy* consumers in relation to the *Benchmark* consumers GD and ZIP. In the moderately competitive settings with 100 providers, only sub-figure 6.15d shows a significant and positive correlation of the CCPAAS and the number of *Q-Strategy* consumers, sub-figure 6.15c does not show any correlation at all.

Figure 6.16 displays the cases with the larger HPC2N job profile. The outcomes in subfigures (a) and (b) confirm the findings of their LLNL equivalences and show significant and positive correlations between the CCPAAS and the number of *Q-*

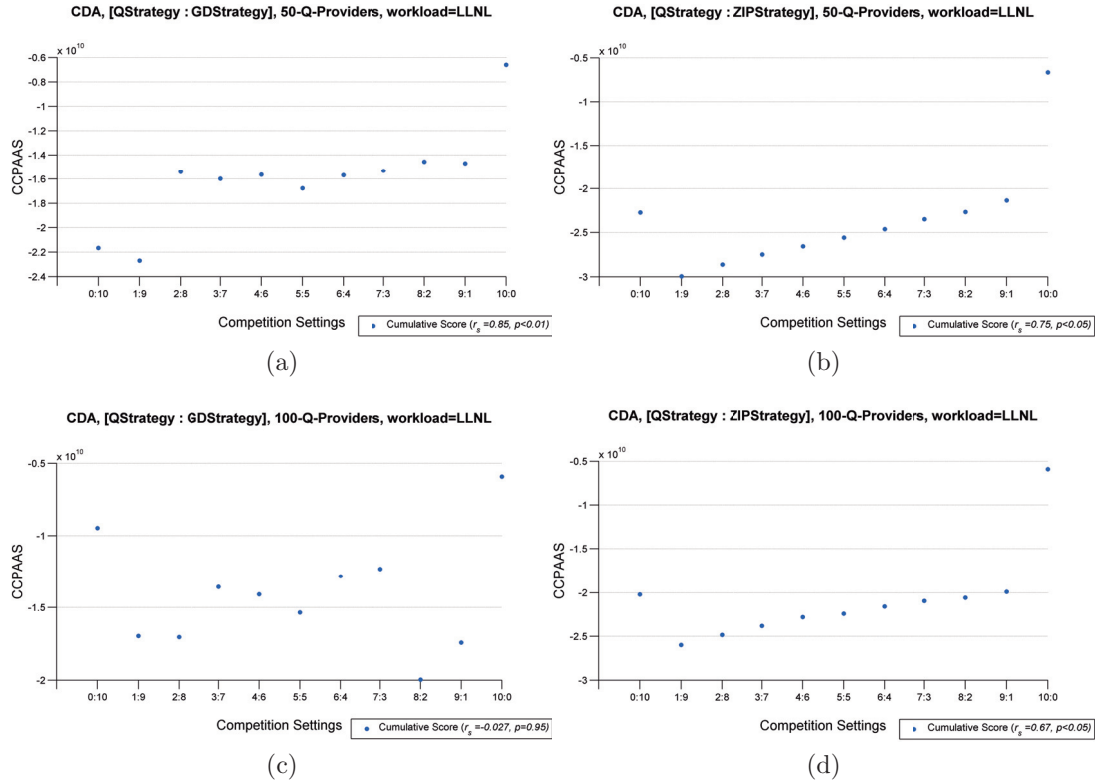


Figure 6.15: Combined consumer and provider outcomes in settings with Q -Providers and the LLNL data profile: (a) Competing settings of (Q -Strategy:GD)-Consumers and 50- Q -Providers; (b) Competing settings of (Q -Strategy:ZIP)-Consumers and 50- Q -Providers; (c) Competing settings of (Q -Strategy:GD)-Consumers and 100- Q -Providers; and, (d) Competing settings of (Q -Strategy:ZIP)-Consumers and 100- Q -Providers.

Strategy consumer agents. Sub-figure (d) shows a tendency to positive correlations of the CCPAAS and the number of Q -Strategy consumer agents, whereas, sub-figure (c) does not show any correlation at all.

The evaluation results show that although the Q -Strategy only adapts from local experience, it was able to outperform the benchmark strategies in most homogeneous and heterogeneous agent settings, from a consumer and combined perspective. Therefore, Q -Strategy offers a promising approach for automating bidding processes in markets with complete and incomplete information. In markets with incomplete

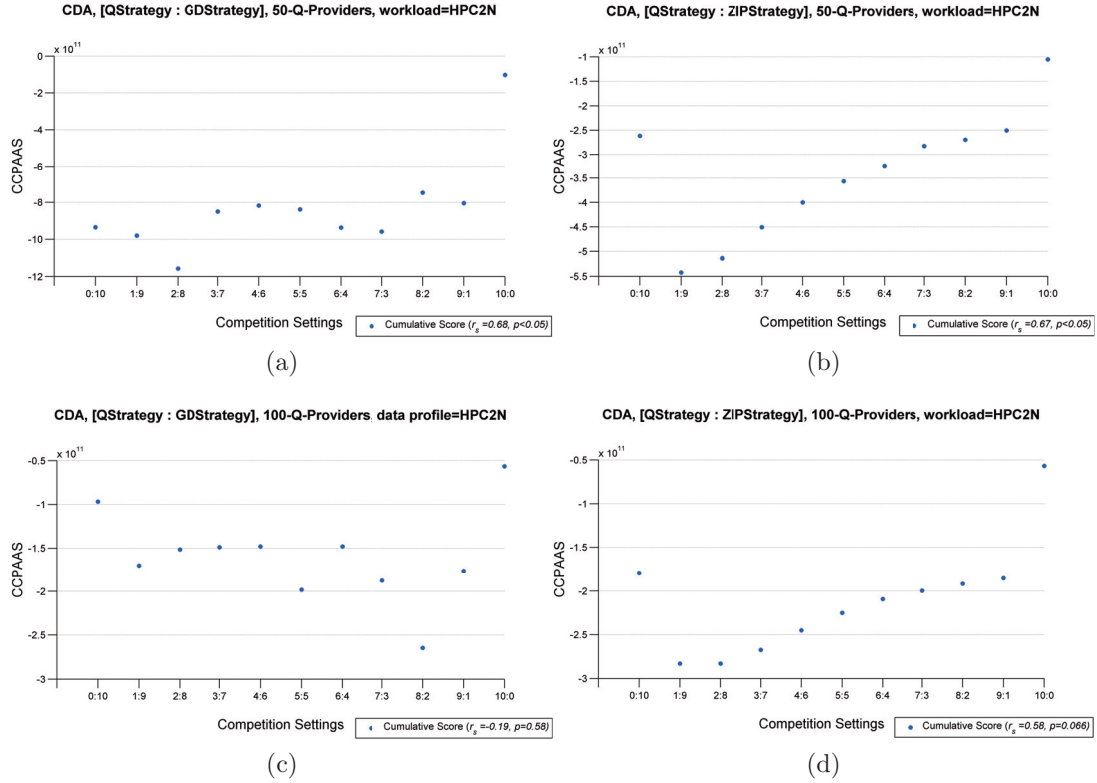


Figure 6.16: Combined consumer and provider outcomes in settings with Q-Providers and the HPC2N data profile: (a) Competing settings of (Q-Strategy:GD)-Consumers and 50-Q-Providers; (b) Competing settings of (Q-Strategy:ZIP)-Consumers and 50-Q-Providers; (c) Competing settings of (Q-Strategy:GD)-Consumers and 100-Q-Providers; and, (d) Competing settings of (Q-Strategy:ZIP)-Consumers and 100-Q-Providers.

information, strategies like ZIP and GD may become infeasible with respect to the public information available, where the *Q-Strategy* offers a good approximation for the exploration and exploitation processes in uncertain environments. However, in the more competitive settings with providers of type ZIP, the price competitive consumer benchmarks ZIP and GD outperformed the *Q-Strategy* consumers.

From a provider's perspective, the price competitive provider strategies ZIP and GD outperformed the *Q-Strategy* providers. However, an interesting result from their outcomes is that all types of providers (ZIP, GD and *Q-Strategy*) profit with an

increasing number of *Q-Strategy* consumers.

Moreover, the evaluation results showed the adaptive and stable behavior of the *Q-Strategy* in the various settings and for both job profiles, LLNL and HPC2N. The combined results of consumer and provider scores confirmed the effectiveness of the *Q-Strategy*. The highest consumer scores and combined scores were achieved in settings with solely *Q-Strategy* providers and *Q-Strategy* consumers as well as in settings, dominated by *Q-Strategy* consumer agents.

6.3 Summary

This chapter presented the economic evaluation of the *Q-Strategy* against the selected (Section 3.3.5) benchmark bidding strategies, ZIP and GD (Section 3.3.4), for the target market scenario of this work: Spot market, implemented with the *Continuous Double Auction* with no time constraints regarding closing times of rounds or days. The applied evaluation methodology is in keeping with existing methodologies of agent-based computational economics and empirical game theoretic analysis. In addition to these methodologies, detailed scenarios with homogenous and heterogeneous settings of bidding strategies for consumers and homogeneous settings for providers have been designed for this work. A detailed description of the evaluation methodology, homogeneous and heterogeneous settings, selected job profiles, the consumer and provider scoring functions, the applied statistical methods and the technical architecture of the agent-based experimental system are presented in Section 6.1. Section 6.2 presented the evidence from the experiments' outcomes from consumer, provider and combined perspectives.

Chapter 7

Technical Analysis and Application

THIS chapter presents the integration of the presented models of this work in three case studies: Two application case studies for batch and interactive jobs and a case study for generating service level agreements. The case studies, *Batch Supply Chain Data Analysis*, *Interactive Sensor Data Analysis* and *e-Service Level Agreements* are part of the SORMA project and realistic candidates for using computing (infrastructure) services on demand. Moreover, the *MX/CS* communication protocol was applied in another application scenario called *Social Cloud* for sharing storage services in social communities.

7.1 Technical Analysis as Methodology

Technical analysis of software artifacts is a commonly applied methodology in design science (Hevner et al., 2004). Hevner et al. (2004) provide guidelines and describe methods for the specification and evaluation of software artifacts. In this context, software artifacts can be semantic constructs (vocabulary and symbols), software engineering models, algorithms and prototypes. The *Information Systems Research Framework* presents the general concepts *Environment*, *IS Research* and *Knowledge Base* for describing and evaluating theories and software artifacts (Hevner et al., 2004). The *Environment* contains a description of the *People* roles, the *Organization* structure, as well as the applied *Technology* in terms of infrastructure and applications. The *IS Research* concept describes the development processes of theories and artifacts and their justification/evaluation through analytical comparison, case studies, experiments, field studies and simulation. The *Knowledge Base* provides the fundamental information for building the theories and artifacts in terms of analyses of existing and commonly applied theories, best practices, existing (formal) frameworks, models and methodologies. This chapter applies the design science methodology by describing the application of the developed models for market-based scheduling in three case studies and in a third-party project. Moreover, a performance analysis

of the integrated SORMA system is performed, which shows the tractability and efficient resource usage of the integrated system components.

7.2 Case Study 1: Batch Supply Chain Data Analysis

7.2.1 TXTDemand and Its Market Scenario

The first application scenario addresses the automated submission and execution of batch jobs on outsourced computing services, which are automatically allocated through a marketplace and according to the supply and demand. Each batch application consists of all required libraries, legal agreements (licenses), input data and configuration files in order to be automatically deployed and executed on the computing services of the allocated external provider.

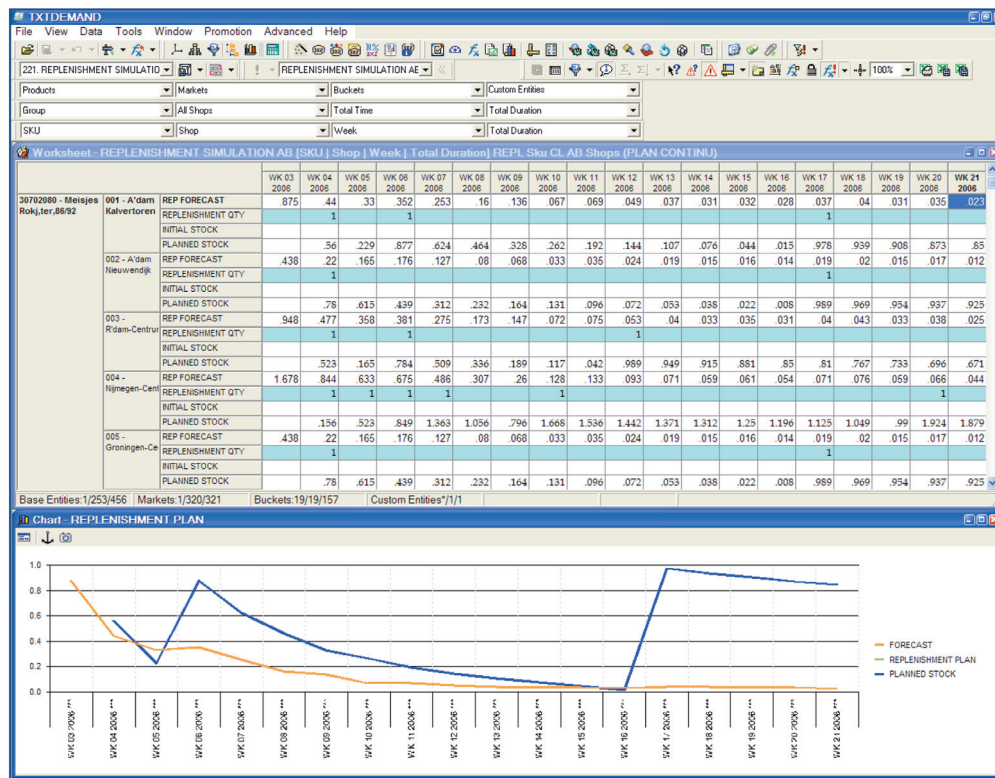


Figure 7.1: TXTDemand: An application for supply chain management and data analysis (screenshot provided by the SORMA project)

TXTDemand is a software application for supply chain management and data analy-

sis, which performs sophisticated calculations using historic data for demand forecasting of future product sales and derivation of replenishment strategies (Windsor et al., 2009; Nimis et al., 2008; Neumann et al., 2007). Figure 7.1 shows the user interface of the *TXTDemand* application. The *TXTDemand* application provider configures and executes the log data of its customers on its local infrastructure. The data logs of the different customers are executed with *TXTDemand* as batch applications during the night since the results often have to be available by the next morning. However, the *TXTDemand* application provider needs a flexible and economically efficient way of acquiring additional computing services on demand to be able to meet customer demand and deal with the changing preferences of customers. Economic efficiency with regard to the *TXTDemand* application provider refers to the platforms, mechanisms, methods, and tools needed to enable a rapid, automatic and market-based allocation of external computing services on demand. The *TXTDemand* provider expects to achieve cost reductions through better utilization of its local infrastructure with acceptable fixed costs by covering its higher demand peaks through the acquisition of computing services from external providers. Moreover, the *TXTDemand* provider expects greater flexibility with regard to license costs and greater customer satisfaction by acquiring differentiable computing services through the market.

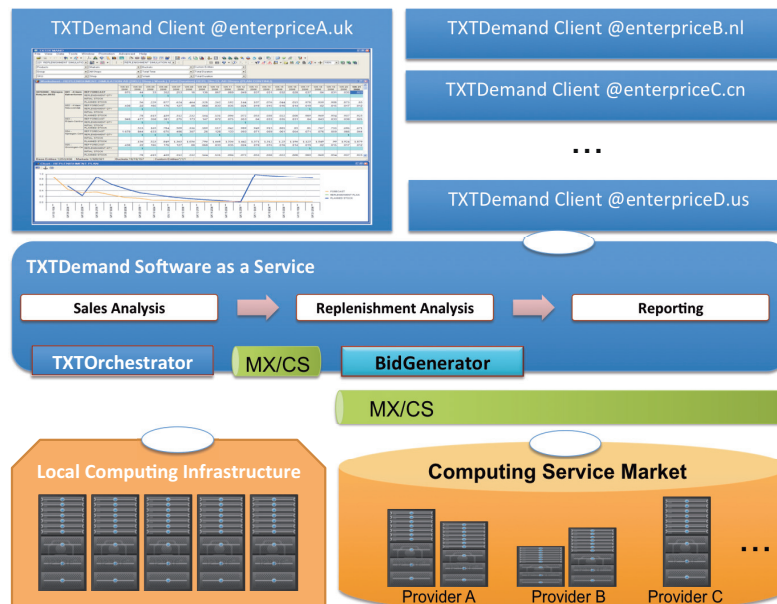


Figure 7.2: Integrated view of *TXTDemand* SaaS, *TXTOrchestrator*, *BidGenerator*, *MX/CS* and the SORMA Market for Computing Services (own representation)

The prototypical integration of *TXTDemand* with the SORMA market is presented in Figure 7.2. According to this representation, all of the *TXTDemand* customers use the *TXTDemand* client in order to view the results of their sales data analysis. In order to receive their results, the customers upload their data to the *TXTDemand Software as a Service*, SaaS application. The *TXTDemand SaaS* performs the steps *Sales Analysis*, *Replenishment Analysis* and *Reporting* according to the customers' preferences and on the *TXTDemand* provider's local infrastructure. If the customer demand exceeds the capacity of the *TXTDemand* provider infrastructure, the *TXTORchestrator* is activated to acquire external computing services from the market.

The *TXTORchestrator* submits a *PrivateMessage* to the owned *BidGenerator*, which starts to execute the bidding processes with the market until an allocation is returned (cf. Figure 4.3 in Section 4.4.2). Based on the received allocation, the *TXTORchestrator* deploys the *TXTDemand* SaaS on the external computing service and starts to execute the customers' sales data analysis.

7.2.2 TXTDemand Requirements

The following *TXTDemand* requirements are summarized from Section 2.4.1.1 and selected according to the models presented for this work:

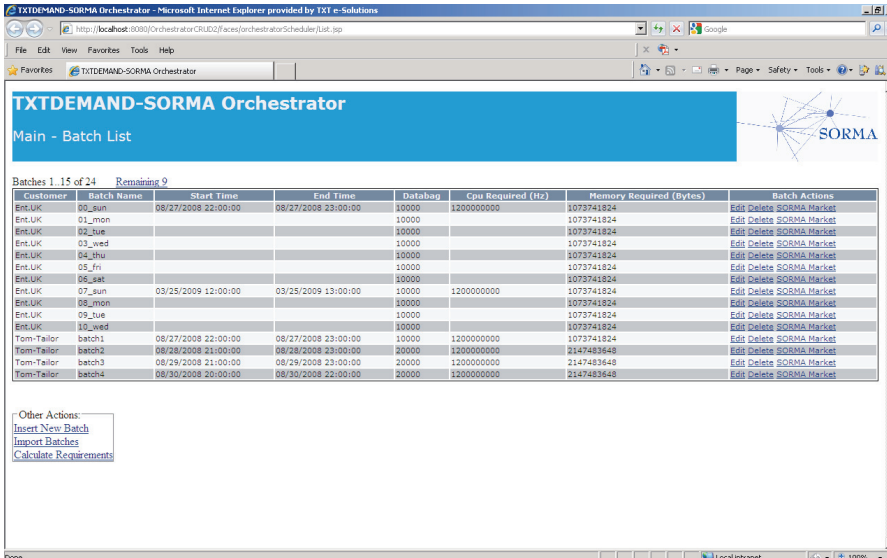
1. The *BidGenerator* should provide clear APIs in order to be adopted by non-experts as well. The *BidGenerator* should support consumers and providers in preparing their bids, and help them understand the economic impact of their actions.
2. Flexibility has to be ensured in case the consumers need more computing services than initially agreed.
3. The transfer of consumer data and bidding processes needs to be fast and performed over a secured communication line, which is part of the SORMA system's communication protocol.
4. Privacy, data protection and security has to be ensured with well-defined policies that are enforced and easy to demonstrate since the *TXTDemand* application handles private and sensitive customer sales data.
5. Negotiations need to be automated and supported according to a clear methodology and service level contracts need to be established for technical parameters

like CPU, memory, storage and bandwidth, as well as for economic parameters like duration and payment.

6. The market-based scheduling system has to support different payment models like pay-per-use, dynamic pricing, prepayments and post-payments.

7.2.3 Integration of BidGenerator and *MX/CS* Protocol

The actual integration of *TXTDemand* with the SORMA market is realized with the *TXTOrchestrator* and its integration with the *BidGenerator*. The *TXTOrchestrator* orchestrates customer batch jobs to computing services allocated from the SORMA market. Figure 7.3 shows the web user interface of the *TXTOrchestrator*. It provides facilities for creating new batch job descriptions, importing batch job descriptions, calculating technical requirements for the batch jobs based on past experience and initialization of bid requests to the *BidGenerator*. The *TXTOrchestrator* web interface shows a list of the uploaded ids for the batch jobs with general technical parameters, time constraints and their values. Based on these parameters, the *TXTOrchestrator* creates a *PrivateMessage* (see Section 5.3.1) with a detailed description of the batch job, the technical requirements and economic preferences.



The screenshot shows the 'Main - Batch List' page of the TXTDemand-SORMA Orchestrator. The page displays a table of batch jobs with the following data:

Customer	Batch Name	Start Time	End Time	Databag	Cpu Required (Hz)	Memory Required (Bytes)	Batch Actions
Ent-UK	02_sun	08/27/2008 22:00:00	08/27/2008 23:00:00	10000	1200000000	1073741824	Edit Delete SORMA Market
Ent-UK	01_mon			10000		1073741824	Edit Delete SORMA Market
Ent-UK	02_tue			10000		1073741824	Edit Delete SORMA Market
Ent-UK	03_wed			10000		1073741824	Edit Delete SORMA Market
Ent-UK	04_thu			10000		1073741824	Edit Delete SORMA Market
Ent-UK	05_fri			10000		1073741824	Edit Delete SORMA Market
Ent-UK	06_sat			10000		1073741824	Edit Delete SORMA Market
Ent-UK	07_sun	03/25/2009 12:00:00	03/25/2009 13:00:00	10000	1200000000	1073741824	Edit Delete SORMA Market
Ent-UK	08_mon			10000		1073741824	Edit Delete SORMA Market
Ent-UK	09_tue			10000		1073741824	Edit Delete SORMA Market
Ent-UK	10_wed			10000		1073741824	Edit Delete SORMA Market
Tom-Tailor	batch1	08/27/2008 22:00:00	08/27/2008 23:00:00	10000	1200000000	1073741824	Edit Delete SORMA Market
Tom-Tailor	batch2	08/29/2008 21:00:00	08/29/2008 23:00:00	20000	1200000000	2147483648	Edit Delete SORMA Market
Tom-Tailor	batch3	08/29/2008 21:00:00	08/29/2008 23:00:00	20000	1200000000	2147483648	Edit Delete SORMA Market
Tom-Tailor	batch4	08/30/2008 20:00:00	08/30/2008 22:00:00	20000	1200000000	2147483648	Edit Delete SORMA Market

Below the table, there are links for 'Other Actions': [Insert New Batch](#), [Import Batches](#), and [Calculate Requirements](#).

Figure 7.3: TXTOrchestrator: General view (screenshot provided by the SORMA project)

Figure 7.4 shows a detailed view of the utilized technical and economic attributes like

time constraints, duration, maximum price (valuation), bidding strategy, payment type and others, which are all part of the *PrivateMessage*. The *TXTOrchestrator* uses the non-blocking Web service interface of *BidGenerator* to submit the *PrivateMessage* for the target batch job. The *BidGenerator* starts the bidding process as specified in the *PrivateMessage* and with the selected bidding strategy. When the bidding process succeeds, the *TXTOrchestrator* receives the *MarketMessage* (Section 5.3.4) back from *BidGenerator* and uses it to deploy and execute the batch job on the target computing service from the allocated external provider.

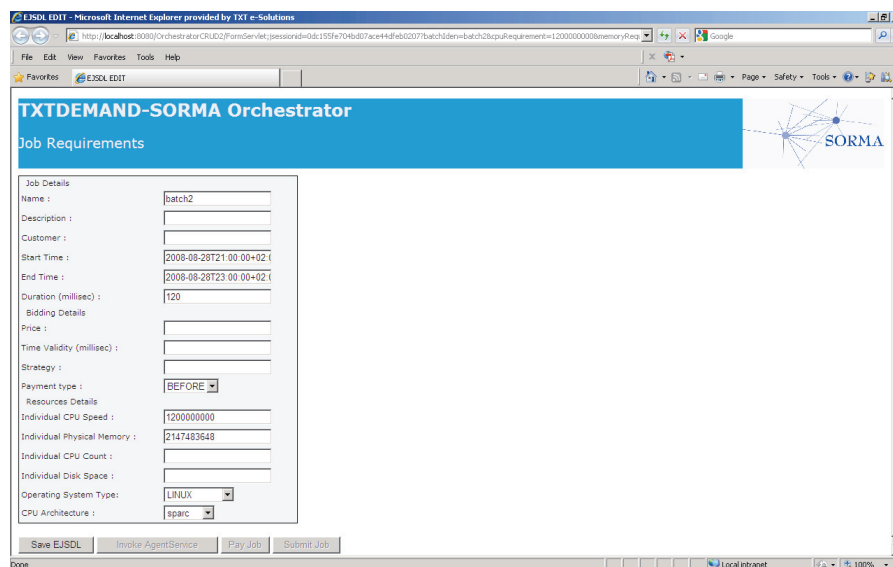


Figure 7.4: TXTOrchestrator: Job details (screenshot provided by the SORMA project)

As part of the project SORMA, *TXTDemand* with *TXTOrchestrator* was integrated, tested and evaluated with *BidGenerator* and the *MX/CS* communication protocol (Figure 7.2). The integrated SORMA prototype was installed on several sites in different countries and continents and its integration was proofed and evaluated by running test jobs using the *TXTDemand* settings. The tests proved end-to-end functionality of the running prototype with distributed, heterogeneous network, hardware and software environments. The *TXTDemand* application evaluation proved the *BidGenerator* and *MX/CS* concepts, but could not give clear evidence on the outcome of the selected bidding strategies because of the heterogeneity of the installed testbeds with different network, hardware and software characteristics, as well as because of the *TXTDemand*-specific design settings for the number of jobs, the number of consumers and the number of providers. Therefore, this work evaluated the bidding

strategies that were part of *BidGenerator*, in a controlled experimental environment with well-defined settings and homogeneous computing nodes of hardware and software configurations, as part of a computing cluster (Chapter 6).

7.2.4 Summary of Case Study Contributions

Requirement 1 of the *TXTDemand* application is addressed in both the architecture and APIs of the *BidGenerator* (Section 4.4), as well as the communication protocol *MX/CS* (Section 5.3). Non-experts can use the *TXTOrchestrator* web interface to specify their technical and economic preferences, which are transformed into a *PrivateMessage*, which is part of the *MX/CS* protocol. The *BidGenerator* uses the values in *PrivateMessage* to initialize the agents with the selected bidding strategy in order to start and execute the bidding processes. Moreover, *BidGenerator* offers clear and simple APIs for experts, who can implement their own agents and bidding strategies, as well as integrate tools for further analysis of the market data. *Requirement 2* is satisfied by the fact that *BidGenerator* can perform multiple negotiations for multiple requests in parallel according to the specified consumer requirements and system resources on which the *BidGenerator* is running. Furthermore, consumers are able to predict and specify their requirements in the *PrivateMessage* and ensure that the executed *TXTDemand* service performs a checkpoint of the actual state and before the requested execution time (job duration) is exceeded. If the job is still not finished, the *TXTOrchestrator* can request a new allocation from the *BidGenerator* and start the already checkpointed job on the new computing service.

Requirement 3 is satisfied by the integrated security mechanism in the *BidGenerator* framework. Each bid generated and submitted to the SORMA system is signed and validated by a trusted identity provider with the SAML protocol. *Requirement 4* addresses security and privacy issues as well. Security and trust in the SORMA system are handled according to the SAML protocol. The security and data protection policies between the application owner and the target computing service provider are also handled with the SAML protocol, but enforced through additional legal and data protection policies by SORMA's *Contract Management*. However, the legal and data privacy aspects have been not investigated in detail in SORMA. However, they will be dealt with in future research.

The automated bidding part of *Requirement 5* is fully supported by *BidGenerator*. The establishment of service level contracts of technical and economic parameters is supported by the *MX/CS* communication protocol and discussed in Section 7.4. *Requirement 6* is addressed in the specification of the *MX/CS* communication protocol,

which incorporates attributes for bids and payment methods like prepayments and post-payments.

7.3 Case Study 2: Interactive Sensor Data Analysis

7.3.1 Visage and Its Market Scenario

The second case study represents the class of interactive web-based applications. *Visage* is an application for analyzing streamed video data (e.g., camera sensors).

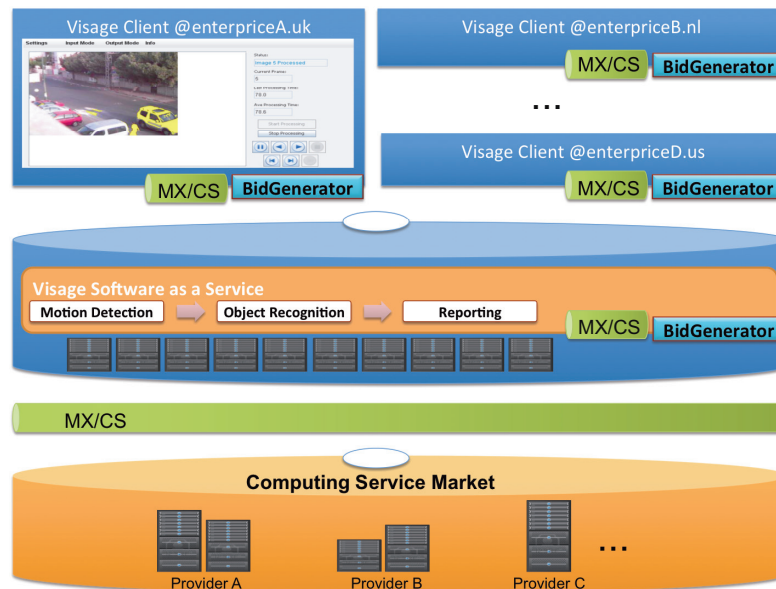


Figure 7.5: Integrated view of TXTDemand SaaS, TXTOrchestrator, BidGenerator, MX/CS and the SORMA Market for Computing Services (own representation)

The data analysis is performed in three logical steps (Figure 7.5). The *Motion Detection* compares a sequence of two or more pictures in order to detect a motion. The *Object Recognition* component performs a pattern recognition to identify the type of moving object. At the end, a *Report* is created and submitted back to the client. *Visage* is provided as an SaaS and invoked by the client. Similar to the previous scenario, the *Visage* provider owns a limited computing infrastructure and aims to cover peak times by acquiring computing services on the market. *Visage clients* invoke the *Visage Service* to get service instances for analyzing the video streams. The *Visage Service* returns instances of the *Visage* provider's infrastructure or if there are no free

capacities, *Visage* invokes *BidGenerator* in order to acquire computing services from the market (cf. Figure 4.3 in Section 4.4.2). On successful allocation, the *Visage* SaaS is deployed on the target computing services of the external providers and the new instances are returned back to the *Visage client*.

7.3.2 Visage Requirements

The following *Visage* requirements are summarized from Section 2.4.1.2 and selected according to the models presented in this work:

1. Automated purchasing and allocation of external computing services from the market according to well-defined and implemented bidding strategies.
2. Efficient deployment of *Visage* on external computing services.
3. *Visage* clients may request several resources simultaneously and the allocations have to take all simultaneous requests into account.
4. The *Visage* system and clients communicate over secured interfaces and all data is protected from unauthorized access.
5. The *Visage* system defines the specific requirements for computing services that have to be taken into account for the bidding processes and fulfilled by the external provider.

The *Visage* requirements are similar to these of *TXTDemand*, however, the interactive applications have different characteristics from the batch applications in terms of technical and economic attributes, security, service level agreement creation and enforcement, as well as the management of contracts and licenses.

7.3.3 Integration of the BidGenerator and MX/CS Protocol

Figure 7.6 presents the integration of the *BidGenerator* and *MX/CS* protocol from the perspective of the *Visage client*. The second part of the *Visage client* shows economic data like *Bidding Strategy* (part of *PrivateMessage*), *Bid* (part of *PublicMessage*), and *Generated Price* (i.e., the *Clearing Price* as part of the returned *MarketMessage*), as part of the *MX/CS* protocol. Moreover, the *MarketMessage* contains the endpoint reference of the target *Visage* service, which is displayed in the *Server url*. In this example, the video data is streamed from a directory and the path

is displayed in the *Video Source* element. The first part of the *Visage client* shows the result of the video data analysis. In the example below, the detected object, the car, is highlighted.

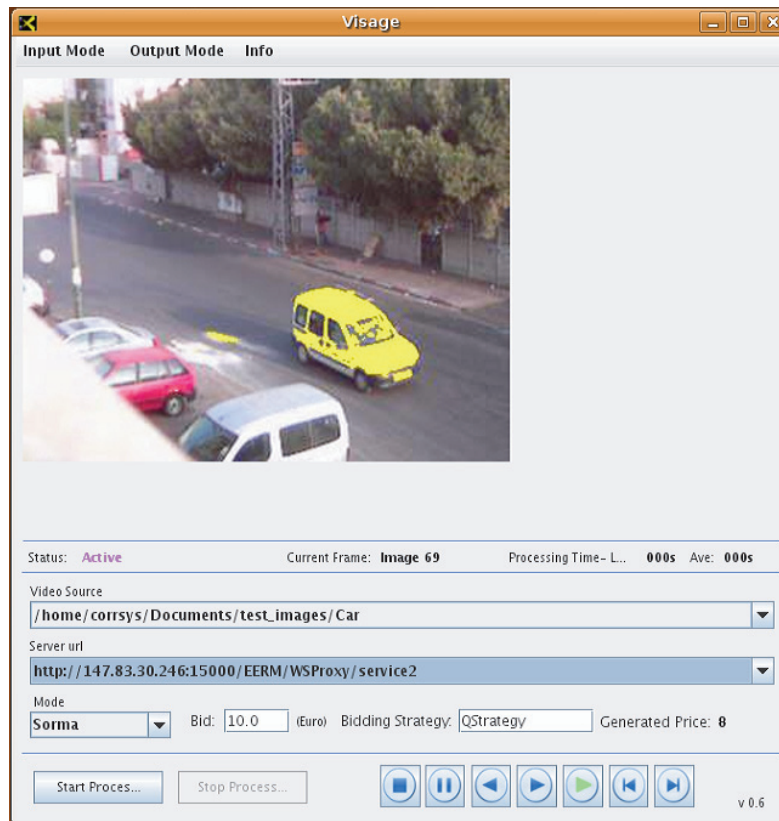


Figure 7.6: Visage client invoked BidGenerator with the Q-Strategy, received an allocation of Visage with an endpoint reference and started an analysis of video data sequences (screenshot provided by the SORMA project)

Figure 7.7¹ shows a real demonstration of the SORMA system with *Visage client*, *Bid-Generator*, *MX/CS* and the SORMA *Resource Manager* called EERM (Macías et al., 2008). The adapted version of both *Visage clients* in this case utilize the *Truth-Telling* bidding strategy when invoking the *BidGenerator* and show aggregated statistics of the already processed images on the different *Visage service* nodes, which have been allocated through *BidGenerator*. The aggregated values are similar because of the

¹Demonstration at IES 2009, Internet of Services 2009, ICT Challenge 1.2 Service and Software Architectures, Infrastructures and Engineering Collaboration meeting for FP6 & FP7 projects.

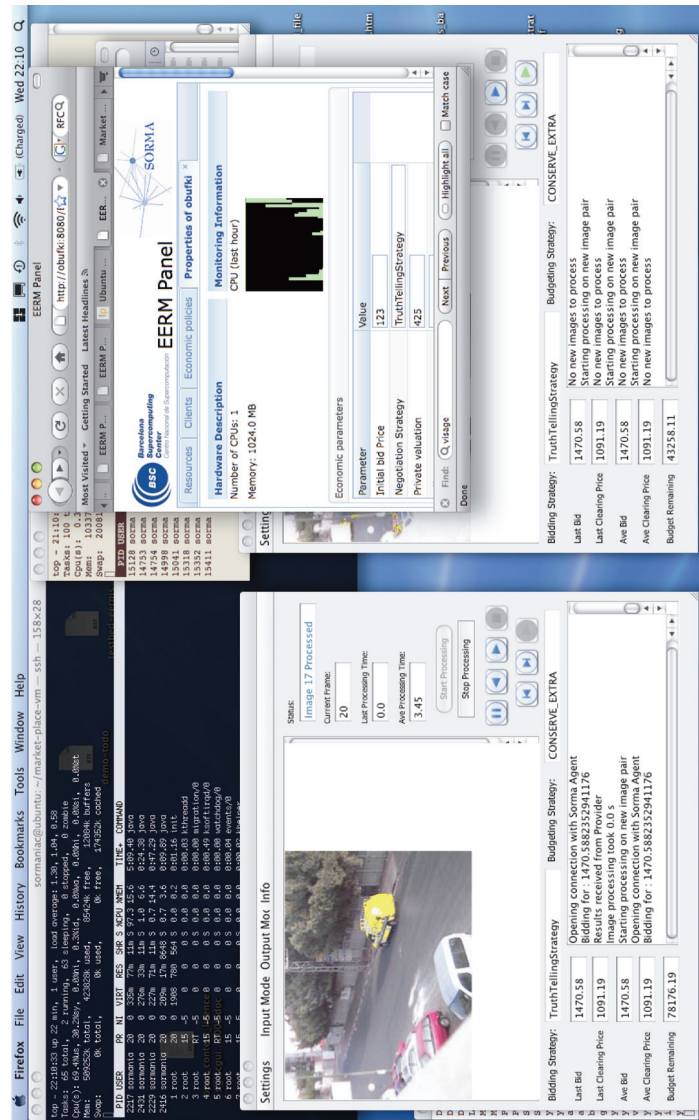


Figure 7.7: Visage integration (Garry Smith, demonstration at IES 2009, project SORMA)

similar economic preferences of the bidding strategy and valuation. The SORMA *Resource Manager* shows monitoring information about the *Visage service* execution (first part of the EERM's web interface), as well as economic information like the bidding strategy and valuation (second part of the EERM's web interface).

Like in *Case Study 1*, the *Visage service* was integrated, tested and evaluated with *BidGenerator* and the *MX/CS* communication protocol as part of the SORMA project and according to the specific settings of the *Visage* scenario.

7.3.4 Summary of Case Study Contributions

Requirement 1 of *Visage* is fully satisfied by *BidGenerator* since it enables automation of the bidding processes. The efficient deployment of *Visage* on external computing services, *Requirement 2*, is enabled by the information provided in *MX/CS* protocol, the contract between the *Visage* owner and the computing service provider. The contract contains the endpoint reference of the target machine, on which the *Visage* SaaS is automatically deployed as a web application in *Tomcat* or *Jetty* through their standard deployment interfaces. *Requirement 3*, simultaneous requests for multiple computing services, is fully supported by the non-blocking interface and scalable.² *BidGenerator* satisfies the part of *Requirement 4*, which addresses its security concept of signing and validating the *MX/CS* messages that are exchanged within the SORMA system. The mechanisms for secured communication between the *Visage client* and deployed *Visage system* are performed outside the SORMA system and *BidGenerator* and are defined and realized by the owner of the *Visage system*.

Requirement 5 is partially satisfied with the *MX/CS* protocol, which provides attributes to specify technical requirements (e.g., by wrapping JSDL) for the system resource configurations like CPU, memory, storage and bandwidth, the endpoint reference for deploying the *Visage service*, as well as the economic preferences for automating bidding processes. However, electronic contracts of software services have to contain application-specific attributes for monitoring and enforcing application-specific key performance indicators. The definition of application-specific attributes was not part of this work, however, it will be a subject area of future research.

²Scalability is limited by the system resources on which *BidGenerator* is running.

7.4 Case Study 3: Application of MX/CS for e-Service Level Agreements

7.4.1 Service Level Agreements in SORMA and Requirements

The result of a matchmaking process in the SORMA system is the creation of a service contract, or a service level agreement (SLA) between the allocated consumer and provider. The SLA includes technical and economic information from the *PublicMessages* of the consumer and provider, which is aggregated according to the matchmaking policy applied (Section 5.4).

The *TXTDemand* and *Visage* scenarios define the following requirements with respect to SLAs (Section 2.4.1):

1. A system for market-based scheduling has to provide the possibility of negotiating negotiate service level agreements and instantiating related contracts. The negotiation process should include technical parameters for raw resources like CPU, memory, storage and bandwidth. A service level agreement has to include information about the duration of a job and the maximum reservation time of the computing service provided.
2. The negotiation protocol has to take into account situations in which resource providers cannot meet the requirements of certain service level agreements and are therefore forced to break them. The economic impact of such events and their causes have to be evaluated and appropriate mechanisms have to be developed to address them (e.g., compensation payments, which are also called penalties).
3. To increase acceptance and trust in the system, providers of computing services have to be evaluated according to well-defined indicators like reliability, performance, etc.
4. A market-based scheduling system has to support different payment models like pay-per-use, dynamic pricing, prepayments and post-payments.
5. The *Visage* system defines specific requirements for computing services, which have to be taken into consideration for the purchasing (bidding) processes. Furthermore, the result of a market-based allocation is a service level agreement, which has to be fulfilled by the external provider.

A *WS-Agreement* is a commonly applied protocol in the Grid community for negotiating service contracts. A *WS-Agreement* defines the general negotiation concepts, a detailed definition of these concepts and their instantiation is left up to the application designer. Moreover, a *WS-Agreement* allows the incorporation of other commonly applied languages like the *Job Submission and Description Language*. In the SORMA context, a *WS-Agreement* is created from the *MarketMessage*, which is the result of the matchmaking process of the consumer's and provider's *PublicMessages*. The *MarketMessage* is transformed into a *WS-Agreement* document, which is also called *MarketMessage* since it incorporates the same data after the syntax-based transformation has taken place. Moreover, a *WS-Agreement* is designed for negotiations and has been adapted through extensions for an auction scenario. The following section describes the creation procedure of a *WS-Agreement* document in SORMA.

7.4.2 Application of the *MX/CS* Protocol

Figure 7.8 illustrates the mapping of the *MarketMessage*³ concepts into a *WS-Agreement* document. A *WS-Agreement* document is created from a *MarketMessage* in six steps. In the *first step*, a globally unique contract identification is generated. In the example below, the ids of the consumer and provider are concatenated with a timestamp. In the *second step*, the consumer and provider ids, together with their bid and offer ids are taken into the *WS-Agreement context*. The *validity period*, which is part of the *context*, specifies the end time of the contract. The *third step* includes the creation of the *WS-Agreement's service properties* – references to the key performance indicators, which are extracted from the technical resource description, which are part of the *MarketMessage*. The *endpoint reference* to the provider's computing service is added in *step four*. In *step five*, the technical description of a *MarketMessage* is taken into the resource description concept of the *WS-Agreement* document. The last and *sixth step* concludes the creation process by extracting the *Guarantee Terms* from the technical description (JSDL) and integrating them into the target concept of the *WS-Agreement* document.

The transformation process of a *MarketMessage* into a *WS-Agreement* document required a definition of the extensions for the latter. For example, concepts like *payment type*, *bid*, *signature* and *clearing price* are missing in the general *WS-Agreement*

³EJSDLMarket is the name of the *MarketMessage*, which uses the JSDL description language for expressing the technical requirements.

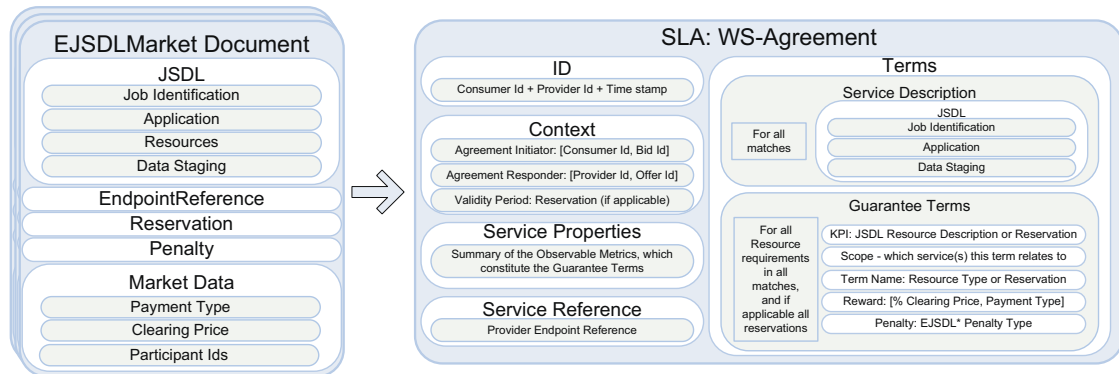


Figure 7.8: Transformation of the MarketMessage concepts into a WS-Agreement (Borissov et al., 2009b).

framework. Moreover, the *WS-Agreement* concepts *Agreement Initiator* and *Agreement Responder* do not fit with the semantics of the mapped *MarketMessage* concepts – *Provider* and *Consumer*. Furthermore, a *WS-Agreement* does not explicitly define the attributes for adding the consumer and provider signatures. These missing attributes have been included as extensions to the *WS-Agreement Guarantee Terms* concept.

Finally, the *WS-Agreement* allows multiple reward and penalty functions with respect to the *Guarantee Terms* to be defined. However, the *WS-Agreement* specifications do not offer any practical suggestions for describing and setting penalty and reward policies, or for mapping and enforcing them. In this context, an incentive compatible penalty function based on *k-pricing* for the market-based scheduling domain was developed by Becker et al. (2008).

7.4.3 Summary of Case Study Contributions

Requirement 1 is addressed in SORMA with the MX/CS protocol and the transformation procedure of *MarketMessage* into a *WS-Agreement* document. *Requirement 2* was not fully addressed in the SORMA system, but supported by the concepts defined for the MX/CS protocol and the research performed in this area (Becker et al., 2008). However, the calculation of penalties and automatic enforcement of service level agreements are emerging subject areas for future research. *Requirement 3* is addressed through the transformation of the *MarketMessage* concepts into *WS-Agreement key performance indicators*, however, more research and evaluation is required in this area. *Requirement 4* is supported by the incorporation of different

payment models and pricing concepts in the definition of an MX/CS protocol. Moreover, these concepts are mapped into the extensions of the *WS-Agreement* document. *Requirement 5* is partially supported by the MX/CS protocol since application services like *Visage* require application-specific attributes, key performance indicators and penalty functions. The Web service URL to the *Visage* SaaS is part of the *endpoint reference* element of the *MX/CS* and *WS-Agreement* documents.

7.5 Further Application Scenarios of MX/CS: Social Cloud

The research project *Social Cloud* elaborates market mechanisms for incentivizing members of social network communities to “share resources amongst each other for little to no gain” (Chard et al., 2011). The computing services can be shared based on monetary, barter or voluntary agreements. In Chard et al. (2011), the *MX/CS* (former EJSDDL) communication protocol was applied for the creation of service level agreements in two market mechanisms, a *Posted Price* and *Reverse Sealed Bid Second Price* auction. Furthermore, the *MX/CS* was also extended by the parameters *availability* and *error rate*, which are used as key performance indicators in the proposed mechanisms. The *Social Cloud* project shows the applicability and extensibility of the *MX/CS* communication protocol with two mechanisms (other than the selected mechanism for this work) for the market-based sharing of storage services through social networks.

7.6 Performance Analysis

7.6.1 Monitoring of Distributed Services

In order to evaluate the performance of a distributed system, access to system and application level information is needed. The target programming language of the SORMA system was Java. Therefore, the components are executed in Java Virtual Machines (JVM). In order to monitor system level or virtual machine level data, the *Java Management Extensions* (JMX) technology can be applied. JMX provides the implementation of managed objects, which is also called MBeans, as well as the tools for managing and monitoring system-level data like CPU utilization, memory usage, storage, threads, devices (e.g., printers) and networks. The JMX provides standard (Web service) interfaces that enable remote access to the monitored target system (JVM).

Figure 7.9 shows system-level information of the monitored *BidGenerator* service

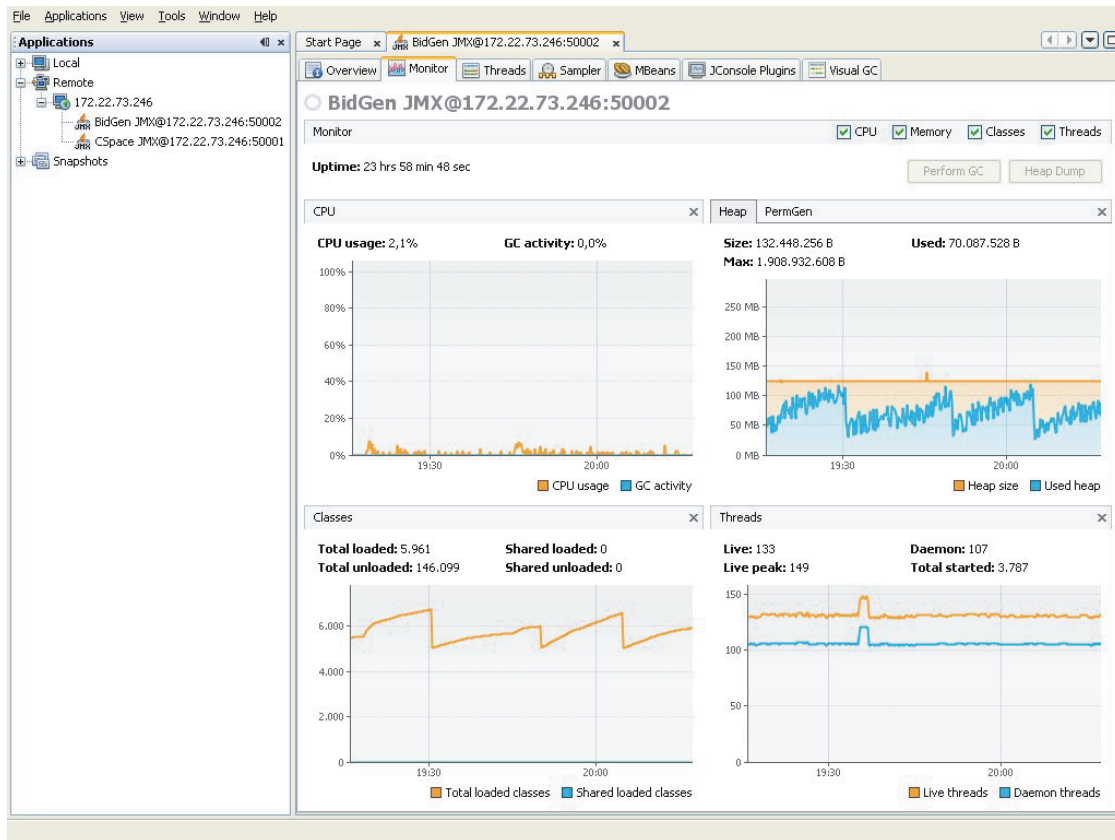


Figure 7.9: Performance Monitor: System specific data with Java's VisualVM (own screenshot)

with Java's VisualVM tool. Here, the *BidGenerator* shows that it performs adequately for the duration of all the experiments, as well as a gentle utilization of the software and hardware resources for the defined settings. However, in settings with a higher number of agents $\#agents \geq 60$ and a high number of parallel bid submissions, over-utilization of the system's resources occurred, which blocked execution of the system. However, achieving scalable systems through decentralization, caching and pre-aggregation techniques is a question of having the hardware resources and respective software configurations.

Monitoring application level data in (distributed) systems can be realized by implementing so-called *System Dashboards*. *System Dashboards* collect and aggregate monitoring information from the distributed system's services. Each of the applications implements a logging mechanism for the internal processes and reports its state

to the remote interface of the *System Dashboard*, as well as according to the specified monitoring data format.

The screenshot displays the SORMA Dashboard web interface. The browser address bar shows the URL `http://ibwgrid.lw.uni-karlsruhe.de:58888/SormaLog/`. The dashboard is divided into several sections:

- Menu:** Includes options for Log4j Server, Log Monitoring, and Bidding.
- Logging Messages:** A scrollable area showing log entries such as:


```
[DEMO]; PROVIDER_ADD; CU; http://magnesium.cs.cf.ac.uk:15000
/EERM; 51.483757, -3.182537; S.J. Caton@cs.cardiff.ac.uk; 0.9.4.2; 2009-10-15 12:28:10,621; (); EERM;;
[DEMO]; PROVIDER_ADD; SICS; http://cluster5.sics.se:48080/EERM; 59.404738, 17.949444; niwi@sics.se; 0.9.4; 2009-10-15 12:19:34,258;
(); EERM;;
[DEMO]; PROVIDER_ADD; CorrSys; http://213.8.173.168:8080
/EERM; 32.029747, 34.856174; g.m.smith@reading.ac.uk; 0.9.4.2; 2009-10-15 12:20:08,336; (); EERM;;
[DEMO]; PROVIDER_ADD; URE; http://portals.acet.rdg.ac.uk:8080
/EERM; 51.437575, -0.941157; g.m.smith@reading.ac.uk; 0.9.4.2; 2009-10-15 12:19:49,156; (); EERM;;
```
- Server State:** A network diagram showing connections between nodes, with the SORMA logo.
- Bidding:** A table with tabs for Bids, Offers, Matches, Rejected Bids, and Rejected Offers. The 'Offers' tab is active, showing:

Valuation (€)	Bid Price (€)	Strategy	Provider Name	Service Type	CPU Architecture	CPU Count	RAM (GB)	Operating System
4.25	12.55	QStrategy	reading	WebService	x86	4	3.46	LINUX
4.25	7.95	QStrategy	bayreuth	WebService	x86	1	0.99	LINUX
- Resources:** A table listing providers with columns for Name, URL, Location, Contact, Version, and Received:

Name	URL	Location	Contact	Version	Received
FZI	http://koralle04.perimeter.fzi	49.011105,8.410688	nimis@fzi.de	0.9.4.2	2009-10-15 12:10:24,549
SICS	http://cluster5.sics.se:48080	59.404738,17.949444	niwi@sics.se	0.9.4	2009-10-15 12:19:34,258
CorrSys	http://213.8.173.168:8080/E	32.029747,34.856174	g.m.smith@reading.ac.uk	0.9.4.2	2009-10-15 12:20:08,336
URE	http://portals.acet.rdg.ac.uk:	51.437575, -0.941157	g.m.smith@reading.ac.uk	0.9.4.2	2009-10-15 12:19:49,156
FZI	http://koralle01.perimeter.fzi	49.011105,8.410688	nimis@fzi.d	0.9.4.2	2009-10-15 12:15:07,992
HUJI	http://mosorma.cs.huji.ac.il:	31.768063,35.215952	g.m.smith@reading.ac.uk	0.9.4.2	2009-10-15 15:19:50,329

Figure 7.10: Distributed service process monitoring: SORMA Dashboard (screenshot provided by the SORMA project)

Figure 7.10 shows the web user interface of the *SORMA Dashboard*. The *SORMA Dashboard* uses *log4j* technology for receiving and aggregating application logs through its remote-access socket appender. In this example, the *SORMA Dashboard* shows logs received from the providers' *Resource Managers*. The registered provider machines are displayed in the panel *Resources*. Moreover, the *Bidding* panel displays the logs received from *BidGenerator* and the SORMA market (*CSpace*), i.e., consumer bids, provider offers, market matches, as well as lists of bids and offers with exceeded time limits that have not been matched.

7.6.2 Technical Performance Experiments

The technical performance of the integrated components is evaluated in several experiments with 10 consumer and 10 provider clients (20 in total), as well as with 20 consumer and 20 provider clients (40 in total). Each of the experiments was executed in several settings – 10 bids per agent up to 500 bids per agent. The metric *makespan* is defined as the time between the submission of the “request for bid” from the application orchestrator, followed by the bid generation processes, until a match is returned back to the application orchestrator. The makespan includes the whole chain of i) *PrivateMessage* creation and submission to *BidGenerator*; ii) the instantiation of the selected bidding strategy in *BidGenerator* followed by the bid generation and creation of a *PublicMessage*, which is submitted to the target market; iii) the matchmaking of the bids in *Cspace*, iv) resulting in the creation of a *MarketMessage*, which is submitted back to the application orchestrator. The selected bidding strategies for the experiments are *Truth-Telling* (simple candidate) and *Q-Strategy* (complex candidate) and the consumer valuations drawn are higher than the provider valuations in order to produce matches for the evaluated number of bids. The experiments have been executed on a 4-core Intel Xeon 3.00GHz processor, the *SORMA market machine* with the installed market components. The *SORMA* market and related components (e.g., *Trusted Market Exchange*, Section 2.4.3) are deployed on individual *Tomcat* servers running within *SORMA market machine* with certain configurations with respect to the number of threads allowed and memory usage. There was a *BidGenerator* for all consumer clients and a *BidGenerator* for all providers’ clients. The test consumers’ and test providers’ clients with their *BidGenerators* are running on two external machines in addition to the *SORMA market machine*, however, in the same network. The aim of this simple experiment was to test the technical scalability and time performance of the integrated *SORMA* system of test clients (simple *Job Orchestrators* and *Resource Orchestrators*), *BidGenerator*, *MX/CS* and *Cspace* in settings with 20 and 40 *Truth-Telling* agents, as well as *Q-Strategy* agents.

With respect to the hardware and software characteristics in terms of number of the possible open connections (constraining the number of running threads), CPU utilization and memory constraints, an artificial time delay between each bid submission per agent was added, which is composed from a constant of 500 milliseconds and additional random milliseconds, which are introduced with the bash’s random number generator $\$RANDOM \in [0, 32767]$. Each of the experiments was repeated 10 times and the results represent the average values. The artificially introduced time delays are selected from a *one-at-a-time* sensitivity analysis and all the experiments

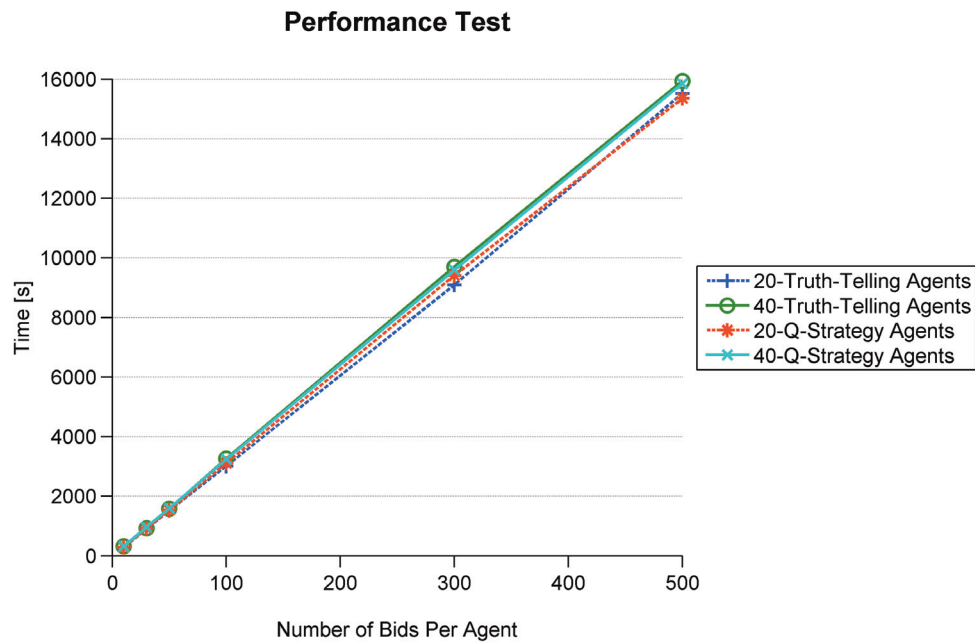


Figure 7.11: Time performance chart of the integrated system in settings with 20 and 40 Truth-Telling agents, as well as Q-Strategy agents

were allowed to be executed without a system block, which typically occurs when a system's capacities are overused and when choosing an artificial delay of below 1000 milliseconds for the largest setting.

Figure 7.11 shows the measured total time (makespan) of the four agent scenarios of 20 and 40 agents that apply either the *Truth-Telling* strategy or *Q-Strategy*. The x-axis represents the number of bids per agents and the y-axis is the total time in seconds needed by all to execute and match their bids. As shown, the makespan increases linearly in relation to the number of bids in all four *Truth-Telling* and *Q-Strategy* agent settings.

Table 7.1 presents the performance results in more detail. The 20 and 40 agents needed an approximate similar makespan (sec) for executing their bids; the linear relation describes the number of executed bids per agent. The largest setting has 40 agents, with each of them submitting 500 bids (i.e., 20000 bids in total), which was executed for an average of 4.45 hours. The smallest setting of 20 agents with 10 bids per agent (i.e., 200 bids in total) was executed for 4.81 minutes. These settings show the linear computational and communication efforts of the proof-of-concept

Table 7.1: Detailed time performance analysis of the integrated system in settings with 20 and 40 Truth-Telling agents, as well as Q-Strategy agents

#bids	<i>Truth-Telling</i>				<i>Q-Strategy</i>			
	#agents (makespan)		Δ (%)	Δ (sec)	#agents (makespan)		Δ (%)	Δ (sec)
20	40	20			40			
10	286	318	11	33	289	320	11	31
30	898	929	3	30	917	964	5	47
50	1524	1581	4	57	1539	1599	4	60
100	3011	3262	8	250	3113	3241	4	127
300	9093	9698	7	605	9408	9593	2	185
500	15511	15937	3	427	15353	15848	3	496

implementation of the components tested, which were constrained by the selected hardware- and software configurations. The Δ parameter shows the time difference between the experiments with 40 and 20 agents expressed in percentage and seconds. These differences increase slowly in the settings with bids of between 10 and 50 and steeper in the remaining ones. The standard deviations of both differences in the *Truth-Telling* strategy settings are 3% and 240 seconds, whereas the the standard deviations for the *Q-Strategy* settings are 3% and 175 seconds, respectively. According to the *Wilcoxon rank sum test*,⁴ the differences in milliseconds between the *Truth-Telling*'s Δ and *Q-Strategy*'s Δ are not significant ($p.value = 0.59$) for the alternative hypothesis $H1: \Delta^{Q-Strategy} > \Delta^{Truth-Telling}$.

7.7 Summary

This chapter presented the technical analysis performed on the models used in this work, *BidGenerator*, *Q-Strategy* and *MX/CS*, for three case studies. The first case study represents the class of batch applications with specific time requirements of customers for the execution and outcome delivery of their data (Section 7.2). In this case, the number and type of batch applications is usually known to the *TXTDemand* application provider a priori, so their executions can be planned ahead. The class of interactive applications represented in the second case study are application services that are executed ad hoc by the consumers without an a priori plan (Section 7.3). The first two case studies show the application of both *BidGenerator* and *MX/CS*. The

⁴A non-parametric statistical hypothesis test used when comparing two related samples or repeated measurements on a single sample to assess whether their population means differ, <<http://stat.ethz.ch/R-manual/R-patched/library/stats/html/wilcox.test.html>>.

third case study shows the application of *MX/CS* for creating e-Service Level Agreements with the *WS-Agreement* specification (Section 7.4). A third-party project, *Social Cloud*, showed the applicability and extensibility of the *MX/CS* communication protocol for two more market mechanisms and in a social networking scenario (Section 7.5). The performance analysis in Section 7.6 showed that the integrated system of *BidGenerator*, *MX/CS* and *CSpace* uses computing services efficiently. In addition, the bidding and communication efforts scale linearly with the number of agents and bids, irrespective of the selected bidding strategy.

Part IV

Conclusion

Chapter 8

Summary of This Work and Future Research

THIS chapter summarizes the key contributions of this work with respect to the research questions introduced in Chapter 1. The final section concludes this thesis with the outlook for future and complementary research.

8.1 Summary of Contributions

The aim of this work was to elaborate, design, develop and realize models for scheduling computing services with market mechanisms. The literature mainly explored the definition and application of market mechanisms in this domain, however, research with regard to the automation of bidding processes in such markets is still largely unexplored. One possible reason is that researchers design market mechanisms that aim to satisfy most of the well-known design desiderata (Myerson and Satterthwaite, 1983). A common solution for incentive compatibility in market design is the application of Vickrey payments. The results of these works and their mathematical models are impressive, however, they are based on idealistic assumptions about the strategic behavior of market participants, e.g., rational bidders that bid truthfully in strategy-proof mechanisms, and informed bidders (all bidders share the same information about the market and other agent's actions). However, such market models are not practical, or computationally and communicationally tractable in real online settings since bidders and their objectives, transaction objects, bidding behavior and timing for bid submission are heterogeneous (Rothkopf, 2007; Wellman et al., 2007). This thesis aims to relax these classic and idealistic assumptions and provide economic models and practical tools to realize adaptive and flexible bidding agents in a pragmatic way. The contributions of this work can be summarized as following:

Q-Strategy. Section 3.4 presented a novel bidding strategy as a model for automated decision making in imperfect market mechanisms with heterogeneous bidders. The *Q-Strategy* is specified and realized for both consumers and providers. In general, the *Q-Strategy* solves the multi-armed bandit problem for the owner's

transaction objects (TO) and for the specified scoring function, which may vary for each of the different TO. A TO is described within the state space of the *Q-Strategy* with its technical and economic attributes. The bids are probabilistically and continuously explored and exploited for each of the different TO in the *Q-Strategy*'s state-action spaces. The adaptive ranking of bids is dynamically updated in the *Q-Strategy*'s Q-Table from the received rewards and for each of the submitted bids. Moreover, the TOs are clustered according to a similarity criterion in order to foster the ranking (learning) processes for the TOs' bids. The *Q-Strategy* adapts fully based on local information and past experience and thus it is applicable in markets with imperfect information, where the dominant bidding strategy is unknown.

BidGenerator Framework. Bidding agents consist of decision making and interactive parts. The decision making part is designed and realized in the bidding strategy module of the BidGenerator framework, the interactive part specifies interfaces to realize agents and their interactions with the owner's applications or computing services, as well as the market. These interfaces implement the *actions* based on an owner's requests, *initialization* of the bidding strategy selected by the owner, the *reaction* to the market messages (e.g., match) and the update of the agent's knowledge base (e.g., the incorporation of the rewards). Therefore, the BidGenerator framework (Section 4.4) offers a platform for developing bidding agents and bidding strategies. The specified interfaces provide a methodology for implementing the core capabilities of bidding agents (Section 4.2). The advantages of decomposing BidGenerator from the target market platform allows consumers and providers to implement, configure and execute their own bidding agents and bidding strategies according to their preferences and needs.

Message Exchange in Computing Service Markets (MX/CS). In order to allocate applications to computing services with market-based schedulers, this work specifies the *MX/CS* communication protocol (Section 5.3). This protocol clearly differentiates between messages that are internal to their owners (called *PrivateMessage*), public messages submitted to the market mechanism (called *PublicMessage*) and match messages (called *MarketMessage*) created as a result of the economic and technical matchmaking processes on the market. Furthermore, the message stack introduces two new message types – *Market-Information* and *StateMessage*, which are applied to query market information

for the bidding strategies, as well as to update their state-action reward information in the agent's knowledge base. The communication protocol is specified as an XML schema and ontology. The consumer, provider and SORMA market components exchange XML messages according to *MX/CS*. The *MX/CS* ontology explicitly defines the concepts and properties of *MX/CS* introduced here. The binding of the XML schema elements and the ontology concepts are implemented with the SAWSDL language. The modular design of the *MX/CS* concepts enable their integration and extensibility with any other standard description language. Here, *MX/CS* was applied on top of the *Job Submission and Description Language*, enhancing its technical attributes with economic modules to support the market-based scheduling processes.

The following part maps the research questions in Chapter 1 to the related contributions of this work.

Research Question 1 <Design of Bidding Strategies>

How can bidding strategies for market-based scheduling be designed and implemented, which when instantiated into bidding agents, automate the bidding process for consumers and providers?

This research question is elaborated in Chapter 3. The chapter contributes to it with an introduction of general design desiderata for developing bidding strategies. The design of a bidding strategy depends on the type of target market mechanism, and its bidding, clearing and information rules. Therefore, the chapter elaborates bidding strategies for settings of perfect information markets and imperfect information markets. Section 3.3.2 presents a general framework for designing bidding strategies for imperfect markets, which was applied to the description of the benchmark bidding strategies and the *Q-Strategy*. As an original contribution to the research, Section 3.4 presents the design and specification of the *Q-Strategy*, which automates the bidding processes for consumers and providers. The *Q-Strategy* was implemented in the Bid-Generator framework and evaluated in agent-based experiments against benchmark bidding strategies in the *Continuous Double Auction*.

Research Question 2 <Design of a Framework for Automated Bidding>

What are the characteristics of bidding agents and how can they coincide with bidding strategies in an agent framework for market-based scheduling?

This research question is elaborated in Chapter 4. The chapter deduces common design desiderata for developing agent frameworks for a market-based scheduling domain. Based on the desiderata, an analytical evaluation of existing agent frameworks is shown in this chapter. The BidGenerator component specifies and implements a reference framework for realizing bidding agents and bidding strategies. Furthermore, as a proof of concept, the BidGenerator implements state-of-the-art bidding strategies and the specified interfaces for developing bidding agents. The assignment of agents to bidding strategies is specified within the consumer and provider preferences and performed dynamically on runtime. The BidGenerator framework has a modular architecture and communicates over interfaces with the related components – applications, resource managers and the market. The BidGenerator framework was systematically developed, integrated and tested as a part of the SORMA project. Moreover, the BidGenerator was integrated with a *Discrete Event Engine* to build a Test Box for evaluating bidding strategies and market mechanisms (Nimis et al., 2009). In this context, the Test Box can be used by other researchers to evaluate their own bidding strategies and market mechanisms. The integration process with other market platforms is facilitated with the communication interfaces provided. Chapter 7 presents the integration of the BidGenerator framework in two case studies for batch and interactive applications. The performance analysis showed that the implemented agents, bidding strategies and communication protocol scale linearly with the number of agents and exchanged messages.

Research Question 3 <Communication Protocols>

What are the characteristics of a message exchange within a market-based scheduling context? How can technical and economic preferences be expressed, communicated and matched between consumers, providers and the market?

The characteristics and specifications of a message infrastructure in a system for market-based scheduling is part of Chapter 5. The first part of this research question is elaborated in Section 5.1, in which core design desiderata for realizing communication protocols are identified for market-based scheduling. The state-of-the-art analysis showed that there are no existing protocols, which satisfy the desiderata as needed. The second part of the research question is answered in Section 5.3, which presents a newly developed communication protocol for expressing technical and economic information on transaction objects (computing services). The *MX/CS* communication protocol was one of the first to enable the market-based scheduling of computing services. It has been developed and applied as part of the SORMA prototype in three case studies (Chapter 7), as well as in a third-party application for sharing storage services in social networks called *Social Cloud* (Chard et al., 2011).

Research Question 4 <Evaluation of Bidding Strategies>

How do learning-based bidding strategies score against benchmark bidding strategies in settings with homogeneous and heterogeneous agents?

This research question is elaborated in Chapter 6. Section 6.1 presents the methodology applied for the evaluation the *Q-Strategy* against the benchmark bidding strategies in homogeneous and heterogeneous settings. The evaluation methodology is consistent with well-known works in the area of agent-based computational economics domain. The evaluation results showed the applicability, adaptability and stable behavior of the *Q-Strategy* in varying heterogeneous and homogeneous settings. In contrast to the related work, the selected consumer scoring function is consistent with the market-based scheduling domain. To reduce the complexity of the evaluation, the definition of the experiments focused on the consumer side. The providers are typically assumed to be profit maximizers. The repeated outcome of the 240 unique agent-based experiments showed that the *Q-Strategy* is competitive in more than 70% of the settings for the consumers by outperforming the selected benchmark bidding strategies in scenarios with real job profiles.

8.2 Future Research

This section concludes this work with the presentation of possible future research directions and complementary research.

8.2.1 Hierarchical Bidding and Transfer Learning

Systems like Google's search engine, AppEngine, Amazon's bookstore and Web services, Facebook and Twitter are highly distributed, interconnected and scalable to thousands to millions of users. To achieve autonomous management of complex systems big problems need to be broken down into smaller ones, solved, and the solutions applied to the bigger ones. In the case of automated bidding, agents have to make different decisions about i) which technical parameters are required for a given application and whether these parameters can be determined for other similar applications, such as, ii) what market to choose, iii) the available information on this market, iv) how to bid, etc. These obvious questions result in multiple parameters, which need to be estimated or learned simultaneously. The concept of hierarchical reinforcement learning is a promising one for specifying multiple objectives in sub-units, whose parameters are adapted independently from each other and used to

maximize future rewards in complex decision making processes (Barto and Mahadevan, 2003). A hierarchical bidding strategy can have market, bidding and transfer learning modules. The market module explores the different markets for computing services available – different providers, service configurations, quality of services, etc. The bidding module explores and exploits optimal bids for the selected market from the market module, and the transfer learning module optimizes technical parameters for actual or newly added applications based on the experience of the application executions and available market information. Transfer learning is another area of promising and ongoing research, which deals with the transfer of gained experience from the execution of one task or application and improving the performance of a similar, but different task or application. Transfer learning addresses the question of how to select an appropriate application for the experience to be transferred to and how to achieve this effectively and autonomously (Taylor and Stone, 2009). In the context of bidding, and given the knowledge base for application types x_1 to x_{20} , the question would be how to estimate an appropriate technical configuration for application type x_{21} or y_1 ?

8.2.2 Automated Bidding for Complex Service Mashups

Another area of future research is the bidding in combinatorial auctions for service mashups. This scenario assumes that such web applications are designed with compatible Web service interfaces of input and output data formats. An application is represented with its technical specification and interface description (WSDL), which can be queried from so-called *green pages* (Bernstein, 1996; Diamantini et al., 2007)). Service mashups are created from two or more web applications that work together, integrate with mappings of their input and output interfaces, and exchange message formats and quality aspects (Papazoglou et al., 2007). Moreover, to simplify the integration of web applications, an explicit definition of the interfaces of the applications is required in terms of (machine readable) the semantic descriptions of their parameters, as well as in relation to other existing parameters (Turner et al., 2003; Cusumano, 2008). XML has succeeded in becoming a well-utilized basement language for describing communication protocols, processes, artifacts and Web service interfaces (Gold et al., 2004). Furthermore, description languages like BPEL and WS-CDL are well applied by modeling the business processes of interconnected services (Weske, 2007). A field of ongoing research is the automated bidding context for complex services called *Service Value Networks*, which investigates methodologies, risks and incentives for combining existing application services to create higher level integrated applications (Blau et al., 2009; Michalk and Blau, 2010).

8.2.3 Design of Flexible Market Platforms

Another research direction is the design and implementation of market platforms that use Lisp- and ruby-based languages for configuring, deploying runtime and executing market mechanisms easily. The Game Description Language (GDL) is a declarative language with a Lisp-based syntax that is used to describe arbitrary games (Thielscher, 2010; Love et al., 2006). This type of platform allows market mechanisms and bidding strategies to be expressed and evaluated with more agility. Similar to the Trading Agent Competition, a TAC Cloud game can be specified to incentivize researchers to implement market mechanisms and bidding strategies for market-based scheduling (Cai et al., 2009). The motivation for specifying a TAC Cloud points to the need for more pragmatic and computationally tractable solutions that can interact with research and industry.

8.2.4 Legal Issues and Matchmaking of Service Level Agreements

A match between binding bids results in a contract; this is a legal and binding agreement between a consumer and a provider that captures the already negotiated technical and economic objectives, which are also called *service level objectives* (SLOs). To ensure regular execution of a contract, a target institution monitors the SLOs continuously to ensure that they are being fulfilled, and is also responsible for calculating final payments or penalties (Becker et al., 2008; Wilkes, 2008; Papazoglou and van den Heuvel, 2007). SLOs may include any quality of service attributes (e.g., “service availability of 99.95%”), technical attributes (e.g., number of CPUs), payment procedures, penalties, or legal stipulations. The enforcement of SLOs increases the trustworthiness of consumers and providers (Kephart and Chess, 2003). Kephart and Chess (2003) also identified the need for effective negotiation and matchmaking algorithms, which provide clear rules and govern the processes that create the final contract between consumers and providers. Similar to online markets, a market information system in a market-based scheduling context can capture and provide information about the reliability and quality of services of providers through reputation systems (Josang et al., 2007).

8.2.5 Information Services for Computing Service Markets

Like financial markets, market information systems (MIS) for computing services can aggregate and provide market data with respect to supply and demand (Brunner et al., 2008). Bidding agents can query market information from the MIS service

and use it in bid generation processes (Borissov et al., 2009a). Furthermore, MIS can provide the functionality of “green pages” and store references to running auctions, as well as descriptions of the transaction object types traded. Such information has to be stored in registries and made findable through a query language according to a given technical specification. Websites like *thecloudmarket.com* and *clouDEX-change.org* already provide such information for humans, but do not offer interfaces for machine agents.

8.2.6 Economic Resource Management

The provider’s local resource managers perform the actual allocation of the received applications to their computing infrastructures. This means that control of the application’s execution and satisfaction of the related service level agreement is part of the provider’s scheduling policy (AuYoung et al., 2006; Macías et al., 2008). The design of the provider’s bidding strategies is closely related to the provider’s local scheduling policies and an important part of the provider’s business model since they affect the happiness (outcome) of the consumers. Defining efficient local scheduling policies for computing services by integrating bidding strategies is an area of ongoing research (Kephart and Walsh, 2004b; AuYoung et al., 2006; Becker et al., 2008; Pueschel and Neumann, 2009; Michalk et al., 2011).

8.2.7 Cloud Application Engineering and Standardization

In a market-based scheduling scenario, the definition and application of common standards and tools is crucial for the practicability, trust and acceptance of such a system. On the one hand, providers agree to apply common APIs and tools when offering their infrastructure services to the market; on the other hand, consumers have to utilize these APIs and tools to prepare their applications for such a scenario. Open standards from non-profit organizations like the *Open Grid Forum*, *Distributed Management Task Force* (DMTF, 2010a), *Open Science Grid* and *Open Cloud Manifesto* offer transparent mechanisms and communication protocols for adopting infrastructure services and reducing lock-in effects for consumers (Nelson, 2009).

8.2.8 Complementary Research

BidGenerator as a development framework for bidding agents and strategies can also be applied for trading Cloud futures and derivatives, as well as in Ad Auctions (Meinl and Blau, 2009; Lahaie et al., 2007). Ad Auctions is a promising research field for

bidding strategies since these auctions are frequently executed each day and can be modeled as repeated games with incomplete information (Lahaie et al., 2007). The *Q-Strategy* is a promising candidate for such kinds of auctions and can be further evaluated as part of future research in the TAC/Ad competition (Jordan and Wellman, 2010; Jordan et al., 2010). Moreover, the *BidGenerator*, bidding strategies and the *MX/CS* communication protocol can be applied in market settings for wireless sensor networks, where mobile agents (smartphones, tablet computers) bid for bandwidth (Bratman et al., 2009).

Appendices

Appendix A

Sensitivity Analysis

According to the *one-at-a-time* sensitivity analysis methodology, each setting varies only one parameter of the overall possible combinations.

Tables A.1 and A.4 show the outcomes of the Q-Strategy sensitivity analysis for homogeneous settings of Q-Strategy consumers and providers with LLNL and HPC2N data profiles. The combination of exploration rate $\epsilon^q = 0.3 \in [0.1; 0.3]$, learning rate $\beta^q = 0.1 \in [0.1; 0.3]$ and discount factor $\gamma^q = 0.9 \in [0.1; 0.9]$ achieved the highest *Total Consumer Score* (TCS) of the aggregated consumer scoring function U_j . The parameter ranges are commonly applied and selected from the respective literature (Whiteson and Stone, 2006; Sun and Peterson, 1999; Even-Dar et al., 2003; Even-Dar and Mansour, 2004). The parameter combination with the highest TCS was initialized for all Q-Strategy agents and remained fixed for the duration of the experiment.

Similarly, Tables A.2 and A.5 present the outcomes of the ZIP sensitivity analysis. The ZIP learning rate $\beta^z = 0.5 \in [0.1; 0.5]$ and the momentum coefficient $\gamma^z = 0.1 \in [0.0; 0.1]$ achieved the highest TCS for the LLNL data profile, $\beta^z = 0.2$ and $\gamma^z = 0.9$ scored highest for the HPC2N data profile. The target value ranges for the ZIP parameters are taken as suggested and applied by Cliff and Bruten (1997). The parameter combination $\beta^z = 0.5$ and $\gamma^z = 0.1$ was selected to initialize the ZIP agents and remained fixed for the duration of the experiment. The ZIP sensitivity analysis showed that settings with $\beta^z = 0.5$, $\gamma^z = 0.1$ or combinations thereof are more likely to achieve higher TCSs than the related combinations of $\beta^z = 0.2$ and $\gamma^z = 0.9$.

Tables A.3 and A.6 display the sensitivity analysis of the GD strategy's beta rate parameter β^g . The outcomes of $\beta^g = 250$ and $\beta^g = 400$ are not significantly different. All GD agents have been initialized with $\beta^g = 250$, also called the *fast* price competitiveness choice, which remains fixed for the duration of the experiment (Gjerstad, 2003). Moreover, the selection of $\beta^g = 250$ is justified due to the higher TCS with the larger HPC2N data profile.

Table A.1: Selection of Q-Strategy's parameters in settings with Q-Strategy consumers and providers with the LLNL data profile.

Strategy	ϵ^q	β^q	γ^q	TCS
QStrategy	0.3	0.1	0.9	-179349326598
QStrategy	0.1	0.1	0.9	-188733048145
QStrategy	0.1	0.3	0.9	-190483060168
QStrategy	0.3	0.3	0.9	-194499799223
QStrategy	0.3	0.1	0.2	-197763518085
QStrategy	0.3	0.1	0.1	-199280734261
QStrategy	0.3	0.1	0.5	-203544555310
QStrategy	0.3	0.3	0.2	-209749749374
QStrategy	0.3	0.3	0.5	-209928051158
QStrategy	0.1	0.1	0.5	-210020950464
QStrategy	0.1	0.1	0.2	-216083986357
QStrategy	0.3	0.3	0.1	-217490521652
QStrategy	0.1	0.3	0.2	-222610889409
QStrategy	0.1	0.3	0.5	-228938118420
QStrategy	0.1	0.1	0.1	-245631045967
QStrategy	0.1	0.3	0.1	-253193066141

Table A.2: Selection of ZIP-Strategy's parameters in settings with ZIP consumers and providers with the LLNL data profile.

Strategy	β^z	γ^z	<i>TCS</i>
ZIP	0.5	0.1	-214128131171
ZIP	0.5	0.2	-216255932497
ZIP	0.4	0.1	-216978086142
ZIP	0.4	0.2	-217822968998
ZIP	0.5	0.5	-222629681798
ZIP	0.3	0.1	-222876890288
ZIP	0.3	0.2	-224962053263
ZIP	0.4	0.5	-226750428720
ZIP	0.2	0.9	-226850161164
ZIP	0.3	0.5	-231266085703
ZIP	0.2	0.1	-231791908754
ZIP	0.2	0.2	-233607673291
ZIP	0.2	0.5	-236747889990
ZIP	0.5	0.9	-247485829207
ZIP	0.4	0.9	-249709032425
ZIP	0.3	0.9	-250636753971

Table A.3: Selection of GD-Strategy's parameters in settings with GD-Strategy consumers and providers with the LLNL data profile.

Strategy	β^g	<i>TCS</i>
GD	400.0	-23007840398990
GD	250.0	-23038336800904

Table A.4: Selection of Q-Strategy's parameters in settings with Q-Strategy consumers and providers with the HPC2N data profile.

Strategy	ϵ^q	β^q	γ^q	TCS
QStrategy	0.3	0.1	0.9	-1233325468563
QStrategy	0.3	0.1	0.5	-1288804606586
QStrategy	0.3	0.1	0.1	-1297339420763
QStrategy	0.3	0.1	0.2	-1311908637690
QStrategy	0.3	0.3	0.9	-1325973901181
QStrategy	0.3	0.3	0.1	-1338159520143
QStrategy	0.3	0.3	0.2	-1368911141138
QStrategy	0.3	0.3	0.5	-1386422273595
QStrategy	0.1	0.1	0.2	-2096264081980
QStrategy	0.1	0.3	0.2	-2164988902096
QStrategy	0.1	0.1	0.9	-2202429703125
QStrategy	0.1	0.3	0.9	-2273741211411
QStrategy	0.1	0.1	0.1	-2279232288606
QStrategy	0.1	0.1	0.5	-2318536586561
QStrategy	0.1	0.3	0.1	-2346200703511
QStrategy	0.1	0.3	0.5	-3218125553768

Table A.5: Selection of ZIP-Strategy's parameters in settings with ZIP consumers and providers with the HPC2N data profile.

Strategy	β^z	γ^z	<i>TCS</i>
ZIP	0.2	0.9	-1265165645908
ZIP	0.4	0.1	-1368204677550
ZIP	0.5	0.1	-1377655155815
ZIP	0.5	0.2	-1383321743681
ZIP	0.3	0.1	-1388478567842
ZIP	0.4	0.2	-1394243560545
ZIP	0.3	0.2	-1410936947078
ZIP	0.5	0.5	-1426500194951
ZIP	0.2	0.1	-1430473953714
ZIP	0.4	0.5	-1440322238509
ZIP	0.3	0.5	-1447431577568
ZIP	0.2	0.2	-1455115025598
ZIP	0.2	0.5	-1459452856239
ZIP	0.3	0.9	-1551589042240
ZIP	0.4	0.9	-1620128983229
ZIP	0.5	0.9	-1623809834826

Table A.6: Selection of GD-Strategy's parameters in settings with GD consumers and providers with the HPC2N data profile.

Strategy	β^g	<i>TCS</i>
GD	250.0	-1351355054627723
GD	400.0	-1351516540965796

Appendix B

Full Tables of the Evaluation Results

B.1 Consumer Outcomes

Table B.1: Consumer outcomes of settings with 50 providers and the LLNL workload. The higher the combined CAAS, the better the outcome of the setting.

No.	LLNL_50_Providers		Combined CAAS($\times 10^5$)
	<i>Consumers</i>	<i>Providers</i>	
1	(10 _Q :0 _{GD})	Q-Providers	-68551
2	(10 _Q :0 _{ZIP})	Q-Providers	-69067
3	(8 _Q :2 _{GD})	Q-Providers	-147806
4	(9 _Q :1 _{GD})	Q-Providers	-149139
5	(7 _Q :3 _{GD})	Q-Providers	-155348
6	(2 _Q :8 _{GD})	Q-Providers	-155786
7	(4 _Q :6 _{GD})	Q-Providers	-158339
8	(6 _Q :4 _{GD})	Q-Providers	-158985
9	(10 _Q :0 _{GD})	GD-Providers	-161261
10	(3 _Q :7 _{GD})	Q-Providers	-161600
11	(0 _Q :10 _{ZIP})	ZIP-Providers	-163205
12	(5 _Q :5 _{GD})	Q-Providers	-169742
13	(0 _Q :10 _{ZIP})	GD-Providers	-190029
14	(9 _Q :1 _{ZIP})	Q-Providers	-216004
15	(0 _Q :10 _{GD})	Q-Providers	-218088
16	(0 _Q :10 _{ZIP})	Q-Providers	-228094
17	(1 _Q :9 _{GD})	Q-Providers	-228525
18	(8 _Q :2 _{ZIP})	Q-Providers	-229104
19	(1 _Q :9 _{ZIP})	ZIP-Providers	-230984
20	(7 _Q :3 _{ZIP})	Q-Providers	-237423
21	(2 _Q :8 _{ZIP})	ZIP-Providers	-244075
22	(6 _Q :4 _{ZIP})	Q-Providers	-248494
23	(5 _Q :5 _{ZIP})	Q-Providers	-257951
24	(3 _Q :7 _{ZIP})	ZIP-Providers	-266514
25	(4 _Q :6 _{ZIP})	Q-Providers	-267779

26	(3 _Q :7 _{ZIP})	Q-Providers	-276604
27	(2 _Q :8 _{ZIP})	Q-Providers	-287472
28	(1 _Q :9 _{ZIP})	GD-Providers	-292435
29	(1 _Q :9 _{ZIP})	Q-Providers	-300680
30	(2 _Q :8 _{ZIP})	GD-Providers	-307262
31	(3 _Q :7 _{ZIP})	GD-Providers	-313886
32	(4 _Q :6 _{ZIP})	ZIP-Providers	-316514
33	(7 _Q :3 _{ZIP})	GD-Providers	-316622
34	(4 _Q :6 _{ZIP})	GD-Providers	-323994
35	(6 _Q :4 _{ZIP})	GD-Providers	-324057
36	(5 _Q :5 _{ZIP})	GD-Providers	-333597
37	(8 _Q :2 _{ZIP})	GD-Providers	-334425
38	(5 _Q :5 _{ZIP})	ZIP-Providers	-369474
39	(0 _Q :10 _{GD})	ZIP-Providers	-438683
40	(9 _Q :1 _{ZIP})	GD-Providers	-465031
41	(6 _Q :4 _{ZIP})	ZIP-Providers	-588712
42	(5 _Q :5 _{GD})	GD-Providers	-646030
43	(2 _Q :8 _{GD})	GD-Providers	-750562
44	(7 _Q :3 _{GD})	GD-Providers	-895138
45	(1 _Q :9 _{GD})	ZIP-Providers	-903971
46	(10 _Q :0 _{ZIP})	GD-Providers	-940491
47	(7 _Q :3 _{ZIP})	ZIP-Providers	-966988
48	(1 _Q :9 _{GD})	GD-Providers	-1059026
49	(2 _Q :8 _{GD})	ZIP-Providers	-1149451
50	(6 _Q :4 _{GD})	GD-Providers	-1207267
51	(3 _Q :7 _{GD})	ZIP-Providers	-1255518
52	(10 _Q :0 _{GD})	ZIP-Providers	-1296764
53	(8 _Q :2 _{ZIP})	ZIP-Providers	-1314218
54	(10 _Q :0 _{ZIP})	ZIP-Providers	-1338354
55	(4 _Q :6 _{GD})	GD-Providers	-1340838
56	(3 _Q :7 _{GD})	GD-Providers	-1417129
57	(4 _Q :6 _{GD})	ZIP-Providers	-1480563
58	(0 _Q :10 _{GD})	GD-Providers	-1586744
59	(5 _Q :5 _{GD})	ZIP-Providers	-1725407
60	(9 _Q :1 _{ZIP})	ZIP-Providers	-1830294
61	(8 _Q :2 _{GD})	ZIP-Providers	-1904263
62	(6 _Q :4 _{GD})	ZIP-Providers	-1915432
63	(7 _Q :3 _{GD})	ZIP-Providers	-2096641
64	(9 _Q :1 _{GD})	ZIP-Providers	-2366724
65	(8 _Q :2 _{GD})	GD-Providers	-5942040

66	(9 _Q :1 _{GD})	GD-Providers	-11099648
----	------------------------------------	--------------	-----------

Table B.2: Consumer outcomes of settings with 100 providers and the LLNL workload. The higher the combined CAAS, the better the outcome of the setting.

No.	LLNL_100_Providers		Combined CAAS($\times 10^5$)
	<i>Consumers</i>	<i>Providers</i>	
1	(10 _Q :0 _{ZIP})	Q-Providers	-60184
2	(10 _Q :0 _{GD})	Q-Providers	-60323
3	(10 _Q :0 _{GD})	ZIP-Providers	-73155
4	(10 _Q :0 _{ZIP})	ZIP-Providers	-73956
5	(0 _Q :10 _{GD})	ZIP-Providers	-85604
6	(0 _Q :10 _{GD})	Q-Providers	-95449
7	(10 _Q :0 _{ZIP})	GD-Providers	-110368
8	(10 _Q :0 _{GD})	GD-Providers	-110553
9	(7 _Q :3 _{GD})	Q-Providers	-124026
10	(6 _Q :4 _{GD})	Q-Providers	-129126
11	(3 _Q :7 _{GD})	Q-Providers	-136576
12	(4 _Q :6 _{GD})	Q-Providers	-141748
13	(2 _Q :8 _{GD})	ZIP-Providers	-146821
14	(1 _Q :9 _{GD})	ZIP-Providers	-150837
15	(3 _Q :7 _{GD})	ZIP-Providers	-153308
16	(5 _Q :5 _{GD})	Q-Providers	-154328
17	(4 _Q :6 _{GD})	ZIP-Providers	-155774
18	(6 _Q :4 _{GD})	ZIP-Providers	-160295
19	(0 _Q :10 _{ZIP})	ZIP-Providers	-160900
20	(5 _Q :5 _{GD})	ZIP-Providers	-169469
21	(1 _Q :9 _{GD})	Q-Providers	-170459
22	(2 _Q :8 _{GD})	Q-Providers	-171239
23	(0 _Q :10 _{ZIP})	GD-Providers	-174562
24	(8 _Q :2 _{GD})	ZIP-Providers	-175243
25	(9 _Q :1 _{GD})	Q-Providers	-175353
26	(0 _Q :10 _{GD})	GD-Providers	-177207
27	(7 _Q :3 _{GD})	ZIP-Providers	-179542
28	(9 _Q :1 _{ZIP})	Q-Providers	-200096
29	(8 _Q :2 _{GD})	Q-Providers	-200764
30	(0 _Q :10 _{ZIP})	Q-Providers	-202447
31	(8 _Q :2 _{ZIP})	Q-Providers	-206671

32	(7 _Q :3 _{ZIP})	Q-Providers	-210332
33	(6 _Q :4 _{ZIP})	ZIP-Providers	-211281
34	(4 _Q :6 _{ZIP})	ZIP-Providers	-211314
35	(3 _Q :7 _{ZIP})	ZIP-Providers	-211368
36	(7 _Q :3 _{ZIP})	ZIP-Providers	-211698
37	(9 _Q :1 _{ZIP})	ZIP-Providers	-211921
38	(5 _Q :5 _{ZIP})	ZIP-Providers	-212104
39	(2 _Q :8 _{ZIP})	ZIP-Providers	-212456
40	(8 _Q :2 _{ZIP})	ZIP-Providers	-213239
41	(1 _Q :9 _{ZIP})	ZIP-Providers	-215568
42	(6 _Q :4 _{ZIP})	Q-Providers	-216548
43	(5 _Q :5 _{ZIP})	Q-Providers	-224802
44	(4 _Q :6 _{ZIP})	Q-Providers	-228521
45	(9 _Q :1 _{GD})	ZIP-Providers	-236838
46	(3 _Q :7 _{ZIP})	Q-Providers	-238517
47	(2 _Q :8 _{ZIP})	Q-Providers	-248712
48	(1 _Q :9 _{ZIP})	Q-Providers	-260289
49	(9 _Q :1 _{ZIP})	GD-Providers	-262761
50	(1 _Q :9 _{ZIP})	GD-Providers	-262990
51	(7 _Q :3 _{ZIP})	GD-Providers	-266602
52	(8 _Q :2 _{ZIP})	GD-Providers	-267406
53	(2 _Q :8 _{ZIP})	GD-Providers	-269743
54	(6 _Q :4 _{ZIP})	GD-Providers	-270321
55	(3 _Q :7 _{ZIP})	GD-Providers	-272609
56	(5 _Q :5 _{ZIP})	GD-Providers	-272917
57	(4 _Q :6 _{ZIP})	GD-Providers	-273087
58	(1 _Q :9 _{GD})	GD-Providers	-283668
59	(2 _Q :8 _{GD})	GD-Providers	-332753
60	(4 _Q :6 _{GD})	GD-Providers	-452748
61	(3 _Q :7 _{GD})	GD-Providers	-457567
62	(5 _Q :5 _{GD})	GD-Providers	-571206
63	(6 _Q :4 _{GD})	GD-Providers	-678564
64	(8 _Q :2 _{GD})	GD-Providers	-1560568
65	(7 _Q :3 _{GD})	GD-Providers	-2025546
66	(9 _Q :1 _{GD})	GD-Providers	-3105745

Table B.3: Consumer outcomes of settings with 50 providers and the HPC2N workload. The higher the combined CAAS, the better the outcome of the setting.

No.	HPC2N_50_Providers		Combined CAAS($\times 10^5$)
	<i>Consumers</i>	<i>Providers</i>	
1	(10 _Q :0 _{GD})	Q-Providers	-1054218
2	(10 _Q :0 _{ZIP})	Q-Providers	-1085099
3	(0 _Q :10 _{ZIP})	GD-Providers	-1703405
4	(0 _Q :10 _{ZIP})	ZIP-Providers	-1945762
5	(10 _Q :0 _{ZIP})	GD-Providers	-1945778
6	(10 _Q :0 _{GD})	GD-Providers	-2200169
7	(9 _Q :1 _{ZIP})	Q-Providers	-2535239
8	(0 _Q :10 _{ZIP})	Q-Providers	-2632936
9	(8 _Q :2 _{ZIP})	Q-Providers	-2725949
10	(7 _Q :3 _{ZIP})	Q-Providers	-2855771
11	(1 _Q :9 _{ZIP})	GD-Providers	-3120306
12	(6 _Q :4 _{ZIP})	Q-Providers	-3263527
13	(2 _Q :8 _{ZIP})	GD-Providers	-3454209
14	(5 _Q :5 _{ZIP})	Q-Providers	-3599000
15	(3 _Q :7 _{ZIP})	GD-Providers	-3708171
16	(9 _Q :1 _{ZIP})	GD-Providers	-3793523
17	(7 _Q :3 _{ZIP})	GD-Providers	-3929615
18	(4 _Q :6 _{ZIP})	GD-Providers	-3960010
19	(4 _Q :6 _{ZIP})	Q-Providers	-4030828
20	(6 _Q :4 _{ZIP})	GD-Providers	-4055351
21	(8 _Q :2 _{ZIP})	GD-Providers	-4167194
22	(3 _Q :7 _{ZIP})	Q-Providers	-4535178
23	(5 _Q :5 _{ZIP})	GD-Providers	-4949095
24	(2 _Q :8 _{ZIP})	Q-Providers	-5160078
25	(0 _Q :10 _{GD})	GD-Providers	-5161114
26	(1 _Q :9 _{ZIP})	Q-Providers	-5446674
27	(1 _Q :9 _{ZIP})	ZIP-Providers	-5693947
28	(2 _Q :8 _{ZIP})	ZIP-Providers	-7218910
29	(8 _Q :2 _{GD})	Q-Providers	-7506981
30	(1 _Q :9 _{GD})	GD-Providers	-7961652
31	(9 _Q :1 _{GD})	Q-Providers	-8083268
32	(4 _Q :6 _{GD})	GD-Providers	-8098325
33	(4 _Q :6 _{GD})	Q-Providers	-8206241
34	(5 _Q :5 _{GD})	Q-Providers	-8411912
35	(3 _Q :7 _{GD})	Q-Providers	-8529931

36	(5 _Q :5 _{GD})	GD-Providers	-8972354
37	(0 _Q :10 _{GD})	Q-Providers	-9382095
38	(6 _Q :4 _{GD})	Q-Providers	-9397618
39	(7 _Q :3 _{GD})	GD-Providers	-9435986
40	(3 _Q :7 _{ZIP})	ZIP-Providers	-9563017
41	(7 _Q :3 _{GD})	Q-Providers	-9615037
42	(1 _Q :9 _{GD})	Q-Providers	-9835239
43	(2 _Q :8 _{GD})	GD-Providers	-9852016
44	(8 _Q :2 _{GD})	GD-Providers	-11131176
45	(2 _Q :8 _{GD})	Q-Providers	-11594640
46	(4 _Q :6 _{ZIP})	ZIP-Providers	-11606811
47	(3 _Q :7 _{GD})	GD-Providers	-11672395
48	(0 _Q :10 _{GD})	ZIP-Providers	-11924207
49	(5 _Q :5 _{ZIP})	ZIP-Providers	-14586371
50	(6 _Q :4 _{GD})	GD-Providers	-15923835
51	(6 _Q :4 _{ZIP})	ZIP-Providers	-18853631
52	(7 _Q :3 _{ZIP})	ZIP-Providers	-22436211
53	(10 _Q :0 _{GD})	ZIP-Providers	-25536325
54	(10 _Q :0 _{ZIP})	ZIP-Providers	-25609324
55	(9 _Q :1 _{GD})	GD-Providers	-27581010
56	(1 _Q :9 _{GD})	ZIP-Providers	-27611378
57	(8 _Q :2 _{ZIP})	ZIP-Providers	-27807847
58	(2 _Q :8 _{GD})	ZIP-Providers	-29238046
59	(3 _Q :7 _{GD})	ZIP-Providers	-30494378
60	(4 _Q :6 _{GD})	ZIP-Providers	-32650643
61	(5 _Q :5 _{GD})	ZIP-Providers	-33656891
62	(9 _Q :1 _{ZIP})	ZIP-Providers	-34552918
63	(6 _Q :4 _{GD})	ZIP-Providers	-37592420
64	(7 _Q :3 _{GD})	ZIP-Providers	-39438291
65	(8 _Q :2 _{GD})	ZIP-Providers	-43640894
66	(9 _Q :1 _{GD})	ZIP-Providers	-46991008

Table B.4: Consumer outcomes of settings with 100 providers and the HPC2N workload. The higher the combined CAAS, the better the outcome of the setting.

No.	HPC2N_100_Providers		Combined CAAS($\times 10^5$)
	<i>Consumers</i>	<i>Providers</i>	
1	(10 _Q :0 _{GD})	Q-Providers	-576447
2	(10 _Q :0 _{ZIP})	Q-Providers	-579618
3	(0 _Q :10 _{GD})	ZIP-Providers	-734462
4	(10 _Q :0 _{GD})	GD-Providers	-903385
5	(0 _Q :10 _{GD})	Q-Providers	-974690
6	(10 _Q :0 _{ZIP})	GD-Providers	-1028515
7	(10 _Q :0 _{ZIP})	ZIP-Providers	-1094188
8	(10 _Q :0 _{GD})	ZIP-Providers	-1107360
9	(0 _Q :10 _{GD})	GD-Providers	-1231504
10	(0 _Q :10 _{ZIP})	ZIP-Providers	-1392967
11	(1 _Q :9 _{GD})	ZIP-Providers	-1471657
12	(4 _Q :6 _{GD})	Q-Providers	-1487436
13	(6 _Q :4 _{GD})	Q-Providers	-1487648
14	(3 _Q :7 _{GD})	Q-Providers	-1494709
15	(0 _Q :10 _{ZIP})	GD-Providers	-1502237
16	(2 _Q :8 _{GD})	Q-Providers	-1522503
17	(2 _Q :8 _{GD})	ZIP-Providers	-1541359
18	(3 _Q :7 _{GD})	ZIP-Providers	-1610839
19	(4 _Q :6 _{GD})	ZIP-Providers	-1670399
20	(1 _Q :9 _{GD})	Q-Providers	-1706291
21	(9 _Q :1 _{GD})	Q-Providers	-1771652
22	(5 _Q :5 _{GD})	ZIP-Providers	-1786840
23	(0 _Q :10 _{ZIP})	Q-Providers	-1799236
24	(2 _Q :8 _{ZIP})	ZIP-Providers	-1848090
25	(3 _Q :7 _{ZIP})	ZIP-Providers	-1854137
26	(1 _Q :9 _{ZIP})	ZIP-Providers	-1862732
27	(9 _Q :1 _{ZIP})	Q-Providers	-1864071
28	(4 _Q :6 _{ZIP})	ZIP-Providers	-1872719
29	(7 _Q :3 _{GD})	Q-Providers	-1887748
30	(6 _Q :4 _{GD})	ZIP-Providers	-1907666
31	(5 _Q :5 _{ZIP})	ZIP-Providers	-1920339
32	(8 _Q :2 _{ZIP})	Q-Providers	-1926107
33	(6 _Q :4 _{ZIP})	ZIP-Providers	-1972506
34	(5 _Q :5 _{GD})	Q-Providers	-1993377
35	(7 _Q :3 _{ZIP})	Q-Providers	-2006344

36	(7 _Q :3 _{ZIP})	ZIP-Providers	-2060957
37	(7 _Q :3 _{GD})	ZIP-Providers	-2062742
38	(6 _Q :4 _{ZIP})	Q-Providers	-2102548
39	(8 _Q :2 _{GD})	ZIP-Providers	-2151623
40	(8 _Q :2 _{ZIP})	ZIP-Providers	-2178212
41	(5 _Q :5 _{ZIP})	Q-Providers	-2261270
42	(1 _Q :9 _{ZIP})	GD-Providers	-2294685
43	(9 _Q :1 _{ZIP})	GD-Providers	-2315432
44	(1 _Q :9 _{GD})	GD-Providers	-2353024
45	(9 _Q :1 _{ZIP})	ZIP-Providers	-2359334
46	(6 _Q :4 _{ZIP})	GD-Providers	-2364299
47	(2 _Q :8 _{ZIP})	GD-Providers	-2368400
48	(8 _Q :2 _{ZIP})	GD-Providers	-2383186
49	(7 _Q :3 _{ZIP})	GD-Providers	-2383867
50	(5 _Q :5 _{ZIP})	GD-Providers	-2384827
51	(3 _Q :7 _{ZIP})	GD-Providers	-2386808
52	(4 _Q :6 _{ZIP})	GD-Providers	-2397811
53	(9 _Q :1 _{GD})	ZIP-Providers	-2436342
54	(4 _Q :6 _{ZIP})	Q-Providers	-2458261
55	(2 _Q :8 _{GD})	GD-Providers	-2609021
56	(8 _Q :2 _{GD})	Q-Providers	-2658497
57	(3 _Q :7 _{ZIP})	Q-Providers	-2683325
58	(1 _Q :9 _{ZIP})	Q-Providers	-2835382
59	(2 _Q :8 _{ZIP})	Q-Providers	-2837506
60	(4 _Q :6 _{GD})	GD-Providers	-3469464
61	(3 _Q :7 _{GD})	GD-Providers	-3511246
62	(5 _Q :5 _{GD})	GD-Providers	-4357397
63	(7 _Q :3 _{GD})	GD-Providers	-4383098
64	(6 _Q :4 _{GD})	GD-Providers	-4446110
65	(8 _Q :2 _{GD})	GD-Providers	-4819245
66	(9 _Q :1 _{GD})	GD-Providers	-9898191

B.2 Provider Outcomes

Table B.5: Provider outcomes of settings with 50 providers and the LLNL workload. The higher the PAAS, the better the outcome of the setting.

No.	LLNL_50_Providers		PAAS($\times 10^5$)
	<i>Consumers</i>	<i>Providers</i>	
1	(10 _Q :0 _{ZIP})	ZIP-Providers	6180
2	(10 _Q :0 _{GD})	ZIP-Providers	6135
3	(9 _Q :1 _{GD})	ZIP-Providers	6088
4	(9 _Q :1 _{ZIP})	ZIP-Providers	6018
5	(7 _Q :3 _{GD})	ZIP-Providers	5977
6	(8 _Q :2 _{GD})	ZIP-Providers	5937
7	(6 _Q :4 _{GD})	ZIP-Providers	5867
8	(5 _Q :5 _{GD})	ZIP-Providers	5841
9	(8 _Q :2 _{ZIP})	ZIP-Providers	5779
10	(4 _Q :6 _{GD})	ZIP-Providers	5712
11	(7 _Q :3 _{ZIP})	ZIP-Providers	5595
12	(3 _Q :7 _{GD})	ZIP-Providers	5523
13	(2 _Q :8 _{GD})	ZIP-Providers	5420
14	(6 _Q :4 _{ZIP})	ZIP-Providers	5170
15	(1 _Q :9 _{GD})	ZIP-Providers	5111
16	(0 _Q :10 _{GD})	ZIP-Providers	4831
17	(10 _Q :0 _{GD})	GD-Providers	4696
18	(10 _Q :0 _{ZIP})	GD-Providers	4693
19	(9 _Q :1 _{GD})	GD-Providers	4686
20	(9 _Q :1 _{ZIP})	GD-Providers	4631
21	(8 _Q :2 _{GD})	GD-Providers	4547
22	(8 _Q :2 _{ZIP})	GD-Providers	4511
23	(7 _Q :3 _{GD})	GD-Providers	4486
24	(6 _Q :4 _{GD})	GD-Providers	4455
25	(5 _Q :5 _{ZIP})	ZIP-Providers	4412
26	(5 _Q :5 _{GD})	GD-Providers	4384
27	(7 _Q :3 _{ZIP})	GD-Providers	4353
28	(4 _Q :6 _{GD})	GD-Providers	4340
29	(3 _Q :7 _{GD})	GD-Providers	4272
30	(2 _Q :8 _{GD})	GD-Providers	4208
31	(6 _Q :4 _{ZIP})	GD-Providers	4197
32	(1 _Q :9 _{GD})	GD-Providers	4147
33	(0 _Q :10 _{GD})	GD-Providers	4049

34	(5 _Q :5 _{ZIP})	GD-Providers	3993
35	(4 _Q :6 _{ZIP})	GD-Providers	3748
36	(4 _Q :6 _{ZIP})	ZIP-Providers	3693
37	(3 _Q :7 _{ZIP})	GD-Providers	3348
38	(2 _Q :8 _{ZIP})	GD-Providers	2866
39	(10 _Q :0 _{GD})	Q-Providers	2509
40	(10 _Q :0 _{ZIP})	Q-Providers	2497
41	(9 _Q :1 _{ZIP})	Q-Providers	2453
42	(9 _Q :1 _{GD})	Q-Providers	2443
43	(8 _Q :2 _{ZIP})	Q-Providers	2439
44	(3 _Q :7 _{ZIP})	ZIP-Providers	2380
45	(7 _Q :3 _{ZIP})	Q-Providers	2339
46	(8 _Q :2 _{GD})	Q-Providers	2326
47	(7 _Q :3 _{GD})	Q-Providers	2235
48	(6 _Q :4 _{ZIP})	Q-Providers	2232
49	(6 _Q :4 _{GD})	Q-Providers	2170
50	(5 _Q :5 _{GD})	Q-Providers	2085
51	(5 _Q :5 _{ZIP})	Q-Providers	2075
52	(1 _Q :9 _{ZIP})	GD-Providers	2063
53	(4 _Q :6 _{GD})	Q-Providers	1978
54	(3 _Q :7 _{GD})	Q-Providers	1889
55	(4 _Q :6 _{ZIP})	Q-Providers	1883
56	(2 _Q :8 _{GD})	Q-Providers	1744
57	(1 _Q :9 _{GD})	Q-Providers	1633
58	(3 _Q :7 _{ZIP})	Q-Providers	1612
59	(0 _Q :10 _{GD})	Q-Providers	1532
60	(2 _Q :8 _{ZIP})	ZIP-Providers	1484
61	(2 _Q :8 _{ZIP})	Q-Providers	1345
62	(1 _Q :9 _{ZIP})	Q-Providers	1096
63	(1 _Q :9 _{ZIP})	ZIP-Providers	811
64	(0 _Q :10 _{ZIP})	Q-Providers	703
65	(0 _Q :10 _{ZIP})	ZIP-Providers	463
66	(0 _Q :10 _{ZIP})	GD-Providers	455

Table B.6: Provider outcomes of settings with 100 providers and the LLNL workload. The higher the PAAS, the better the outcome of the setting.

No.	LLNL_100_Providers		PAAS($\times 10^5$)
	<i>Consumers</i>	<i>Providers</i>	
1	(10 _Q :0 _{GD})	GD-Providers	2095
2	(10 _Q :0 _{ZIP})	GD-Providers	2079
3	(9 _Q :1 _{ZIP})	GD-Providers	2029
4	(9 _Q :1 _{GD})	GD-Providers	2023
5	(8 _Q :2 _{GD})	GD-Providers	1953
6	(8 _Q :2 _{ZIP})	GD-Providers	1934
7	(7 _Q :3 _{GD})	GD-Providers	1887
8	(7 _Q :3 _{ZIP})	GD-Providers	1833
9	(6 _Q :4 _{GD})	GD-Providers	1804
10	(5 _Q :5 _{GD})	GD-Providers	1734
11	(6 _Q :4 _{ZIP})	GD-Providers	1732
12	(4 _Q :6 _{GD})	GD-Providers	1627
13	(5 _Q :5 _{ZIP})	GD-Providers	1592
14	(3 _Q :7 _{GD})	GD-Providers	1522
15	(4 _Q :6 _{ZIP})	GD-Providers	1456
16	(2 _Q :8 _{GD})	GD-Providers	1380
17	(10 _Q :0 _{ZIP})	ZIP-Providers	1309
18	(10 _Q :0 _{GD})	ZIP-Providers	1306
19	(3 _Q :7 _{ZIP})	GD-Providers	1268
20	(9 _Q :1 _{GD})	ZIP-Providers	1228
21	(1 _Q :9 _{GD})	GD-Providers	1223
22	(9 _Q :1 _{ZIP})	ZIP-Providers	1155
23	(10 _Q :0 _{GD})	Q-Providers	1154
24	(8 _Q :2 _{GD})	ZIP-Providers	1136
25	(10 _Q :0 _{ZIP})	Q-Providers	1133
26	(9 _Q :1 _{GD})	Q-Providers	1096
27	(0 _Q :10 _{GD})	GD-Providers	1089
28	(7 _Q :3 _{GD})	ZIP-Providers	1077
29	(8 _Q :2 _{GD})	Q-Providers	1054
30	(9 _Q :1 _{ZIP})	Q-Providers	1045
31	(2 _Q :8 _{ZIP})	GD-Providers	1023
32	(6 _Q :4 _{GD})	ZIP-Providers	1023
33	(8 _Q :2 _{ZIP})	ZIP-Providers	999
34	(7 _Q :3 _{GD})	Q-Providers	992
35	(5 _Q :5 _{GD})	ZIP-Providers	973

36	(6 _Q :4 _{GD})	Q-Providers	958
37	(8 _Q :2 _{ZIP})	Q-Providers	937
38	(4 _Q :6 _{GD})	ZIP-Providers	919
39	(5 _Q :5 _{GD})	Q-Providers	913
40	(7 _Q :3 _{ZIP})	ZIP-Providers	894
41	(4 _Q :6 _{GD})	Q-Providers	891
42	(7 _Q :3 _{ZIP})	Q-Providers	862
43	(3 _Q :7 _{GD})	ZIP-Providers	861
44	(3 _Q :7 _{GD})	Q-Providers	838
45	(2 _Q :8 _{GD})	Q-Providers	817
46	(2 _Q :8 _{GD})	ZIP-Providers	810
47	(6 _Q :4 _{ZIP})	ZIP-Providers	776
48	(1 _Q :9 _{GD})	Q-Providers	773
49	(1 _Q :9 _{GD})	ZIP-Providers	767
50	(6 _Q :4 _{ZIP})	Q-Providers	767
51	(0 _Q :10 _{GD})	Q-Providers	745
52	(0 _Q :10 _{GD})	ZIP-Providers	715
53	(5 _Q :5 _{ZIP})	Q-Providers	698
54	(1 _Q :9 _{ZIP})	GD-Providers	687
55	(5 _Q :5 _{ZIP})	ZIP-Providers	658
56	(4 _Q :6 _{ZIP})	Q-Providers	600
57	(4 _Q :6 _{ZIP})	ZIP-Providers	548
58	(3 _Q :7 _{ZIP})	Q-Providers	524
59	(2 _Q :8 _{ZIP})	Q-Providers	451
60	(3 _Q :7 _{ZIP})	ZIP-Providers	446
61	(1 _Q :9 _{ZIP})	Q-Providers	362
62	(2 _Q :8 _{ZIP})	ZIP-Providers	340
63	(0 _Q :10 _{ZIP})	Q-Providers	259
64	(1 _Q :9 _{ZIP})	ZIP-Providers	229
65	(0 _Q :10 _{ZIP})	GD-Providers	191
66	(0 _Q :10 _{ZIP})	ZIP-Providers	103

Table B.7: Provider outcomes of settings with 50 providers and the HPC2N workload. The higher the PAAS, the better the outcome of the setting.

No.	HPC2N_50_Providers		PAAS($\times 10^5$)
	<i>Consumers</i>	<i>Providers</i>	
1	(10 _Q :0 _{GD})	ZIP-Providers	53314
2	(10 _Q :0 _{ZIP})	ZIP-Providers	53066
3	(9 _Q :1 _{GD})	ZIP-Providers	52757
4	(8 _Q :2 _{GD})	ZIP-Providers	52172
5	(9 _Q :1 _{ZIP})	ZIP-Providers	51785
6	(7 _Q :3 _{GD})	ZIP-Providers	51670
7	(6 _Q :4 _{GD})	ZIP-Providers	50804
8	(8 _Q :2 _{ZIP})	ZIP-Providers	50281
9	(5 _Q :5 _{GD})	ZIP-Providers	49697
10	(4 _Q :6 _{GD})	ZIP-Providers	49222
11	(7 _Q :3 _{ZIP})	ZIP-Providers	48436
12	(3 _Q :7 _{GD})	ZIP-Providers	47866
13	(2 _Q :8 _{GD})	ZIP-Providers	47313
14	(6 _Q :4 _{ZIP})	ZIP-Providers	45952
15	(1 _Q :9 _{GD})	ZIP-Providers	45884
16	(0 _Q :10 _{GD})	ZIP-Providers	44638
17	(5 _Q :5 _{ZIP})	ZIP-Providers	42491
18	(10 _Q :0 _{GD})	GD-Providers	41716
19	(10 _Q :0 _{ZIP})	GD-Providers	41680
20	(9 _Q :1 _{GD})	GD-Providers	41223
21	(9 _Q :1 _{ZIP})	GD-Providers	41084
22	(8 _Q :2 _{GD})	GD-Providers	40748
23	(8 _Q :2 _{ZIP})	GD-Providers	40363
24	(7 _Q :3 _{GD})	GD-Providers	40253
25	(6 _Q :4 _{GD})	GD-Providers	39759
26	(7 _Q :3 _{ZIP})	GD-Providers	39507
27	(5 _Q :5 _{GD})	GD-Providers	39309
28	(4 _Q :6 _{ZIP})	ZIP-Providers	38602
29	(4 _Q :6 _{GD})	GD-Providers	38574
30	(6 _Q :4 _{ZIP})	GD-Providers	38218
31	(3 _Q :7 _{GD})	GD-Providers	37929
32	(2 _Q :8 _{GD})	GD-Providers	37191
33	(5 _Q :5 _{ZIP})	GD-Providers	37086
34	(1 _Q :9 _{GD})	GD-Providers	36290
35	(4 _Q :6 _{ZIP})	GD-Providers	34993

36	(0 _Q :10 _{GD})	GD-Providers	34616
37	(3 _Q :7 _{ZIP})	ZIP-Providers	34321
38	(3 _Q :7 _{ZIP})	GD-Providers	32327
39	(10 _Q :0 _{GD})	Q-Providers	32011
40	(10 _Q :0 _{ZIP})	Q-Providers	31997
41	(9 _Q :1 _{ZIP})	Q-Providers	31691
42	(8 _Q :2 _{ZIP})	Q-Providers	31415
43	(9 _Q :1 _{GD})	Q-Providers	31382
44	(8 _Q :2 _{GD})	Q-Providers	31125
45	(7 _Q :3 _{ZIP})	Q-Providers	31002
46	(6 _Q :4 _{ZIP})	Q-Providers	30650
47	(7 _Q :3 _{GD})	Q-Providers	30596
48	(5 _Q :5 _{ZIP})	Q-Providers	30143
49	(6 _Q :4 _{GD})	Q-Providers	30038
50	(5 _Q :5 _{GD})	Q-Providers	29700
51	(4 _Q :6 _{GD})	Q-Providers	29191
52	(4 _Q :6 _{ZIP})	Q-Providers	29168
53	(3 _Q :7 _{GD})	Q-Providers	28565
54	(2 _Q :8 _{ZIP})	GD-Providers	28032
55	(2 _Q :8 _{GD})	Q-Providers	27881
56	(3 _Q :7 _{ZIP})	Q-Providers	27845
57	(2 _Q :8 _{ZIP})	ZIP-Providers	27493
58	(1 _Q :9 _{GD})	Q-Providers	26894
59	(0 _Q :10 _{GD})	Q-Providers	26244
60	(2 _Q :8 _{ZIP})	Q-Providers	25831
61	(1 _Q :9 _{ZIP})	Q-Providers	23127
62	(1 _Q :9 _{ZIP})	GD-Providers	21709
63	(1 _Q :9 _{ZIP})	ZIP-Providers	20589
64	(0 _Q :10 _{ZIP})	Q-Providers	17394
65	(0 _Q :10 _{ZIP})	ZIP-Providers	10038
66	(0 _Q :10 _{ZIP})	GD-Providers	7714

Table B.8: Provider outcomes of settings with 100 providers and the HPC2N workload. The higher the PAAS, the better the outcome of the setting.

No.	HPC2N_100_Providers		PAAS($\times 10^5$)
	<i>Consumers</i>	<i>Providers</i>	
1	(10 _Q :0 _{GD})	GD-Providers	17669
2	(10 _Q :0 _{ZIP})	GD-Providers	17619
3	(9 _Q :1 _{GD})	GD-Providers	17122
4	(9 _Q :1 _{ZIP})	GD-Providers	17049
5	(8 _Q :2 _{GD})	GD-Providers	16593
6	(8 _Q :2 _{ZIP})	GD-Providers	16408
7	(7 _Q :3 _{GD})	GD-Providers	16139
8	(7 _Q :3 _{ZIP})	GD-Providers	15631
9	(6 _Q :4 _{GD})	GD-Providers	15459
10	(5 _Q :5 _{GD})	GD-Providers	14843
11	(6 _Q :4 _{ZIP})	GD-Providers	14761
12	(4 _Q :6 _{GD})	GD-Providers	14095
13	(10 _Q :0 _{GD})	ZIP-Providers	14074
14	(10 _Q :0 _{ZIP})	ZIP-Providers	14062
15	(5 _Q :5 _{ZIP})	GD-Providers	13787
16	(9 _Q :1 _{GD})	ZIP-Providers	13371
17	(3 _Q :7 _{GD})	GD-Providers	13322
18	(9 _Q :1 _{ZIP})	ZIP-Providers	12817
19	(4 _Q :6 _{ZIP})	GD-Providers	12543
20	(8 _Q :2 _{GD})	ZIP-Providers	12468
21	(2 _Q :8 _{GD})	GD-Providers	12227
22	(7 _Q :3 _{GD})	ZIP-Providers	11910
23	(6 _Q :4 _{GD})	ZIP-Providers	11147
24	(1 _Q :9 _{GD})	GD-Providers	11117
25	(8 _Q :2 _{ZIP})	ZIP-Providers	11041
26	(3 _Q :7 _{ZIP})	GD-Providers	10997
27	(10 _Q :0 _{GD})	Q-Providers	10821
28	(10 _Q :0 _{ZIP})	Q-Providers	10793
29	(9 _Q :1 _{ZIP})	Q-Providers	10511
30	(9 _Q :1 _{GD})	Q-Providers	10461
31	(5 _Q :5 _{GD})	ZIP-Providers	10308
32	(8 _Q :2 _{ZIP})	Q-Providers	10178
33	(8 _Q :2 _{GD})	Q-Providers	10125
34	(7 _Q :3 _{ZIP})	Q-Providers	9908
35	(0 _Q :10 _{GD})	GD-Providers	9838

36	(7 _Q :3 _{GD})	Q-Providers	9791
37	(4 _Q :6 _{GD})	ZIP-Providers	9682
38	(7 _Q :3 _{ZIP})	ZIP-Providers	9650
39	(6 _Q :4 _{ZIP})	Q-Providers	9575
40	(6 _Q :4 _{GD})	Q-Providers	9413
41	(5 _Q :5 _{ZIP})	Q-Providers	9210
42	(3 _Q :7 _{GD})	ZIP-Providers	9098
43	(5 _Q :5 _{GD})	Q-Providers	9057
44	(2 _Q :8 _{ZIP})	GD-Providers	8990
45	(4 _Q :6 _{GD})	Q-Providers	8745
46	(4 _Q :6 _{ZIP})	Q-Providers	8734
47	(2 _Q :8 _{GD})	ZIP-Providers	8499
48	(3 _Q :7 _{GD})	Q-Providers	8313
49	(6 _Q :4 _{ZIP})	ZIP-Providers	8124
50	(3 _Q :7 _{ZIP})	Q-Providers	8022
51	(1 _Q :9 _{GD})	ZIP-Providers	7989
52	(2 _Q :8 _{GD})	Q-Providers	7950
53	(0 _Q :10 _{GD})	ZIP-Providers	7494
54	(1 _Q :9 _{GD})	Q-Providers	7474
55	(2 _Q :8 _{ZIP})	Q-Providers	7079
56	(0 _Q :10 _{GD})	Q-Providers	6990
57	(5 _Q :5 _{ZIP})	ZIP-Providers	6840
58	(1 _Q :9 _{ZIP})	GD-Providers	6043
59	(1 _Q :9 _{ZIP})	Q-Providers	5712
60	(4 _Q :6 _{ZIP})	ZIP-Providers	5496
61	(3 _Q :7 _{ZIP})	ZIP-Providers	4179
62	(0 _Q :10 _{ZIP})	Q-Providers	3075
63	(2 _Q :8 _{ZIP})	ZIP-Providers	2902
64	(1 _Q :9 _{ZIP})	ZIP-Providers	1834
65	(0 _Q :10 _{ZIP})	GD-Providers	1278
66	(0 _Q :10 _{ZIP})	ZIP-Providers	845

B.3 Combined Outcomes

Table B.9: Combined outcomes of settings with 50 providers and the LLNL workload. The higher the CCPAAS, the better the outcome of the setting.

No.	LLNL_50.Providers		CCPAAS($\times 10^5$)
	<i>Consumers</i>	<i>Providers</i>	
1	(10 _Q :0 _{GD})	Q-Providers	-66042
2	(10 _Q :0 _{ZIP})	Q-Providers	-66570
3	(8 _Q :2 _{GD})	Q-Providers	-145479
4	(9 _Q :1 _{GD})	Q-Providers	-146696
5	(7 _Q :3 _{GD})	Q-Providers	-153113
6	(2 _Q :8 _{GD})	Q-Providers	-154041
7	(4 _Q :6 _{GD})	Q-Providers	-156360
8	(10 _Q :0 _{GD})	GD-Providers	-156565
9	(6 _Q :4 _{GD})	Q-Providers	-156814
10	(3 _Q :7 _{GD})	Q-Providers	-159711
11	(0 _Q :10 _{ZIP})	ZIP-Providers	-162742
12	(5 _Q :5 _{GD})	Q-Providers	-167657
13	(0 _Q :10 _{ZIP})	GD-Providers	-189574
14	(9 _Q :1 _{ZIP})	Q-Providers	-213552
15	(0 _Q :10 _{GD})	Q-Providers	-216556
16	(8 _Q :2 _{ZIP})	Q-Providers	-226665
17	(1 _Q :9 _{GD})	Q-Providers	-226892
18	(0 _Q :10 _{ZIP})	Q-Providers	-227391
19	(1 _Q :9 _{ZIP})	ZIP-Providers	-230174
20	(7 _Q :3 _{ZIP})	Q-Providers	-235083
21	(2 _Q :8 _{ZIP})	ZIP-Providers	-242591
22	(6 _Q :4 _{ZIP})	Q-Providers	-246261
23	(5 _Q :5 _{ZIP})	Q-Providers	-255876
24	(3 _Q :7 _{ZIP})	ZIP-Providers	-264133
25	(4 _Q :6 _{ZIP})	Q-Providers	-265897
26	(3 _Q :7 _{ZIP})	Q-Providers	-274992
27	(2 _Q :8 _{ZIP})	Q-Providers	-286127
28	(1 _Q :9 _{ZIP})	GD-Providers	-290372
29	(1 _Q :9 _{ZIP})	Q-Providers	-299583
30	(2 _Q :8 _{ZIP})	GD-Providers	-304396
31	(3 _Q :7 _{ZIP})	GD-Providers	-310538
32	(7 _Q :3 _{ZIP})	GD-Providers	-312269
33	(4 _Q :6 _{ZIP})	ZIP-Providers	-312821

34	(6 _Q :4 _{ZIP})	GD-Providers	-319860
35	(4 _Q :6 _{ZIP})	GD-Providers	-320246
36	(5 _Q :5 _{ZIP})	GD-Providers	-329604
37	(8 _Q :2 _{ZIP})	GD-Providers	-329914
38	(5 _Q :5 _{ZIP})	ZIP-Providers	-365061
39	(0 _Q :10 _{GD})	ZIP-Providers	-433852
40	(9 _Q :1 _{ZIP})	GD-Providers	-460400
41	(6 _Q :4 _{ZIP})	ZIP-Providers	-583542
42	(5 _Q :5 _{GD})	GD-Providers	-641646
43	(2 _Q :8 _{GD})	GD-Providers	-746354
44	(7 _Q :3 _{GD})	GD-Providers	-890652
45	(1 _Q :9 _{GD})	ZIP-Providers	-898860
46	(10 _Q :0 _{ZIP})	GD-Providers	-935798
47	(7 _Q :3 _{ZIP})	ZIP-Providers	-961393
48	(1 _Q :9 _{GD})	GD-Providers	-1054879
49	(2 _Q :8 _{GD})	ZIP-Providers	-1144031
50	(6 _Q :4 _{GD})	GD-Providers	-1202812
51	(3 _Q :7 _{GD})	ZIP-Providers	-1249995
52	(10 _Q :0 _{GD})	ZIP-Providers	-1290629
53	(8 _Q :2 _{ZIP})	ZIP-Providers	-1308439
54	(10 _Q :0 _{ZIP})	ZIP-Providers	-1332174
55	(4 _Q :6 _{GD})	GD-Providers	-1336498
56	(3 _Q :7 _{GD})	GD-Providers	-1412856
57	(4 _Q :6 _{GD})	ZIP-Providers	-1474851
58	(0 _Q :10 _{GD})	GD-Providers	-1582696
59	(5 _Q :5 _{GD})	ZIP-Providers	-1719566
60	(9 _Q :1 _{ZIP})	ZIP-Providers	-1824275
61	(8 _Q :2 _{GD})	ZIP-Providers	-1898326
62	(6 _Q :4 _{GD})	ZIP-Providers	-1909565
63	(7 _Q :3 _{GD})	ZIP-Providers	-2090664
64	(9 _Q :1 _{GD})	ZIP-Providers	-2360636
65	(8 _Q :2 _{GD})	GD-Providers	-5937493
66	(9 _Q :1 _{GD})	GD-Providers	-11094962

Table B.10: Combined outcomes of settings with 100 providers and the LLNL workload. The higher the CCPAAS, the better the outcome of the setting.

No.	LLNL_100_Providers		CCPAAS($\times 10^5$)
	<i>Consumers</i>	<i>Providers</i>	
1	(10 _Q :0 _{ZIP})	Q-Providers	-59051
2	(10 _Q :0 _{GD})	Q-Providers	-59169
3	(10 _Q :0 _{GD})	ZIP-Providers	-71849
4	(10 _Q :0 _{ZIP})	ZIP-Providers	-72647
5	(0 _Q :10 _{GD})	ZIP-Providers	-84889
6	(0 _Q :10 _{GD})	Q-Providers	-94704
7	(10 _Q :0 _{ZIP})	GD-Providers	-108289
8	(10 _Q :0 _{GD})	GD-Providers	-108458
9	(7 _Q :3 _{GD})	Q-Providers	-123034
10	(6 _Q :4 _{GD})	Q-Providers	-128168
11	(3 _Q :7 _{GD})	Q-Providers	-135739
12	(4 _Q :6 _{GD})	Q-Providers	-140857
13	(2 _Q :8 _{GD})	ZIP-Providers	-146011
14	(1 _Q :9 _{GD})	ZIP-Providers	-150070
15	(3 _Q :7 _{GD})	ZIP-Providers	-152447
16	(5 _Q :5 _{GD})	Q-Providers	-153415
17	(4 _Q :6 _{GD})	ZIP-Providers	-154855
18	(6 _Q :4 _{GD})	ZIP-Providers	-159272
19	(0 _Q :10 _{ZIP})	ZIP-Providers	-160796
20	(5 _Q :5 _{GD})	ZIP-Providers	-168496
21	(1 _Q :9 _{GD})	Q-Providers	-169686
22	(2 _Q :8 _{GD})	Q-Providers	-170422
23	(8 _Q :2 _{GD})	ZIP-Providers	-174107
24	(9 _Q :1 _{GD})	Q-Providers	-174257
25	(0 _Q :10 _{ZIP})	GD-Providers	-174371
26	(0 _Q :10 _{GD})	GD-Providers	-176118
27	(7 _Q :3 _{GD})	ZIP-Providers	-178465
28	(9 _Q :1 _{ZIP})	Q-Providers	-199050
29	(8 _Q :2 _{GD})	Q-Providers	-199710
30	(0 _Q :10 _{ZIP})	Q-Providers	-202188
31	(8 _Q :2 _{ZIP})	Q-Providers	-205734
32	(7 _Q :3 _{ZIP})	Q-Providers	-209471
33	(6 _Q :4 _{ZIP})	ZIP-Providers	-210505
34	(9 _Q :1 _{ZIP})	ZIP-Providers	-210765
35	(4 _Q :6 _{ZIP})	ZIP-Providers	-210766

36	(7 _Q :3 _{ZIP})	ZIP-Providers	-210804
37	(3 _Q :7 _{ZIP})	ZIP-Providers	-210922
38	(5 _Q :5 _{ZIP})	ZIP-Providers	-211446
39	(2 _Q :8 _{ZIP})	ZIP-Providers	-212116
40	(8 _Q :2 _{ZIP})	ZIP-Providers	-212240
41	(1 _Q :9 _{ZIP})	ZIP-Providers	-215338
42	(6 _Q :4 _{ZIP})	Q-Providers	-215781
43	(5 _Q :5 _{ZIP})	Q-Providers	-224103
44	(4 _Q :6 _{ZIP})	Q-Providers	-227921
45	(9 _Q :1 _{GD})	ZIP-Providers	-235611
46	(3 _Q :7 _{ZIP})	Q-Providers	-237993
47	(2 _Q :8 _{ZIP})	Q-Providers	-248261
48	(1 _Q :9 _{ZIP})	Q-Providers	-259927
49	(9 _Q :1 _{ZIP})	GD-Providers	-260732
50	(1 _Q :9 _{ZIP})	GD-Providers	-262303
51	(7 _Q :3 _{ZIP})	GD-Providers	-264769
52	(8 _Q :2 _{ZIP})	GD-Providers	-265473
53	(6 _Q :4 _{ZIP})	GD-Providers	-268589
54	(2 _Q :8 _{ZIP})	GD-Providers	-268719
55	(5 _Q :5 _{ZIP})	GD-Providers	-271325
56	(3 _Q :7 _{ZIP})	GD-Providers	-271341
57	(4 _Q :6 _{ZIP})	GD-Providers	-271631
58	(1 _Q :9 _{GD})	GD-Providers	-282445
59	(2 _Q :8 _{GD})	GD-Providers	-331373
60	(4 _Q :6 _{GD})	GD-Providers	-451121
61	(3 _Q :7 _{GD})	GD-Providers	-456045
62	(5 _Q :5 _{GD})	GD-Providers	-569471
63	(6 _Q :4 _{GD})	GD-Providers	-676760
64	(8 _Q :2 _{GD})	GD-Providers	-1558615
65	(7 _Q :3 _{GD})	GD-Providers	-2023660
66	(9 _Q :1 _{GD})	GD-Providers	-3103722

Table B.11: Combined outcomes of settings with 50 providers and the LLNL workload. The higher the CCPAAS, the better the outcome of the setting.

No.	HPC2N_50_Providers		CCPAAS($\times 10^5$)
	<i>Consumers</i>	<i>Providers</i>	
1	(10 _Q :0 _{GD})	Q-Providers	-1022207
2	(10 _Q :0 _{ZIP})	Q-Providers	-1053102
3	(0 _Q :10 _{ZIP})	GD-Providers	-1695691
4	(10 _Q :0 _{ZIP})	GD-Providers	-1904098
5	(0 _Q :10 _{ZIP})	ZIP-Providers	-1935723
6	(10 _Q :0 _{GD})	GD-Providers	-2158453
7	(9 _Q :1 _{ZIP})	Q-Providers	-2503549
8	(0 _Q :10 _{ZIP})	Q-Providers	-2615542
9	(8 _Q :2 _{ZIP})	Q-Providers	-2694534
10	(7 _Q :3 _{ZIP})	Q-Providers	-2824769
11	(1 _Q :9 _{ZIP})	GD-Providers	-3098597
12	(6 _Q :4 _{ZIP})	Q-Providers	-3232877
13	(2 _Q :8 _{ZIP})	GD-Providers	-3426177
14	(5 _Q :5 _{ZIP})	Q-Providers	-3568857
15	(3 _Q :7 _{ZIP})	GD-Providers	-3675844
16	(9 _Q :1 _{ZIP})	GD-Providers	-3752439
17	(7 _Q :3 _{ZIP})	GD-Providers	-3890107
18	(4 _Q :6 _{ZIP})	GD-Providers	-3925017
19	(4 _Q :6 _{ZIP})	Q-Providers	-4001660
20	(6 _Q :4 _{ZIP})	GD-Providers	-4017133
21	(8 _Q :2 _{ZIP})	GD-Providers	-4126832
22	(3 _Q :7 _{ZIP})	Q-Providers	-4507333
23	(5 _Q :5 _{ZIP})	GD-Providers	-4912009
24	(0 _Q :10 _{GD})	GD-Providers	-5126498
25	(2 _Q :8 _{ZIP})	Q-Providers	-5134247
26	(1 _Q :9 _{ZIP})	Q-Providers	-5423547
27	(1 _Q :9 _{ZIP})	ZIP-Providers	-5673358
28	(2 _Q :8 _{ZIP})	ZIP-Providers	-7191417
29	(8 _Q :2 _{GD})	Q-Providers	-7475856
30	(1 _Q :9 _{GD})	GD-Providers	-7925362
31	(9 _Q :1 _{GD})	Q-Providers	-8051885
32	(4 _Q :6 _{GD})	GD-Providers	-8059751
33	(4 _Q :6 _{GD})	Q-Providers	-8177051
34	(5 _Q :5 _{GD})	Q-Providers	-8382212
35	(3 _Q :7 _{GD})	Q-Providers	-8501367

36	(5 _Q :5 _{GD})	GD-Providers	-8933045
37	(0 _Q :10 _{GD})	Q-Providers	-9355851
38	(6 _Q :4 _{GD})	Q-Providers	-9367580
39	(7 _Q :3 _{GD})	GD-Providers	-9395733
40	(3 _Q :7 _{ZIP})	ZIP-Providers	-9528696
41	(7 _Q :3 _{GD})	Q-Providers	-9584441
42	(1 _Q :9 _{GD})	Q-Providers	-9808346
43	(2 _Q :8 _{GD})	GD-Providers	-9814825
44	(8 _Q :2 _{GD})	GD-Providers	-11090428
45	(2 _Q :8 _{GD})	Q-Providers	-11566760
46	(4 _Q :6 _{ZIP})	ZIP-Providers	-11568209
47	(3 _Q :7 _{GD})	GD-Providers	-11634466
48	(0 _Q :10 _{GD})	ZIP-Providers	-11879569
49	(5 _Q :5 _{ZIP})	ZIP-Providers	-14543880
50	(6 _Q :4 _{GD})	GD-Providers	-15884076
51	(6 _Q :4 _{ZIP})	ZIP-Providers	-18807679
52	(7 _Q :3 _{ZIP})	ZIP-Providers	-22387774
53	(10 _Q :0 _{GD})	ZIP-Providers	-25483011
54	(10 _Q :0 _{ZIP})	ZIP-Providers	-25556257
55	(9 _Q :1 _{GD})	GD-Providers	-27539786
56	(1 _Q :9 _{GD})	ZIP-Providers	-27565494
57	(8 _Q :2 _{ZIP})	ZIP-Providers	-27757566
58	(2 _Q :8 _{GD})	ZIP-Providers	-29190732
59	(3 _Q :7 _{GD})	ZIP-Providers	-30446512
60	(4 _Q :6 _{GD})	ZIP-Providers	-32601421
61	(5 _Q :5 _{GD})	ZIP-Providers	-33607194
62	(9 _Q :1 _{ZIP})	ZIP-Providers	-34501134
63	(6 _Q :4 _{GD})	ZIP-Providers	-37541616
64	(7 _Q :3 _{GD})	ZIP-Providers	-39386621
65	(8 _Q :2 _{GD})	ZIP-Providers	-43588722
66	(9 _Q :1 _{GD})	ZIP-Providers	-46938252

Table B.12: Combined outcomes of settings with 50 providers and the LLNL workload. The higher the CCPAAS, the better the outcome of the setting.

No.	HPC2N_100_Providers		CCPAAS($\times 10^5$)
	<i>Consumers</i>	<i>Providers</i>	
1	(10 _Q :0 _{GD})	Q-Providers	-565626
2	(10 _Q :0 _{ZIP})	Q-Providers	-568826
3	(0 _Q :10 _{GD})	ZIP-Providers	-726968
4	(10 _Q :0 _{GD})	GD-Providers	-885716
5	(0 _Q :10 _{GD})	Q-Providers	-967701
6	(10 _Q :0 _{ZIP})	GD-Providers	-1010896
7	(10 _Q :0 _{ZIP})	ZIP-Providers	-1080126
8	(10 _Q :0 _{GD})	ZIP-Providers	-1093286
9	(0 _Q :10 _{GD})	GD-Providers	-1221666
10	(0 _Q :10 _{ZIP})	ZIP-Providers	-1392122
11	(1 _Q :9 _{GD})	ZIP-Providers	-1463668
12	(6 _Q :4 _{GD})	Q-Providers	-1478235
13	(4 _Q :6 _{GD})	Q-Providers	-1478691
14	(3 _Q :7 _{GD})	Q-Providers	-1486396
15	(0 _Q :10 _{ZIP})	GD-Providers	-1500959
16	(2 _Q :8 _{GD})	Q-Providers	-1514553
17	(2 _Q :8 _{GD})	ZIP-Providers	-1532860
18	(3 _Q :7 _{GD})	ZIP-Providers	-1601741
19	(4 _Q :6 _{GD})	ZIP-Providers	-1660717
20	(1 _Q :9 _{GD})	Q-Providers	-1698818
21	(9 _Q :1 _{GD})	Q-Providers	-1761191
22	(5 _Q :5 _{GD})	ZIP-Providers	-1776532
23	(0 _Q :10 _{ZIP})	Q-Providers	-1796161
24	(2 _Q :8 _{ZIP})	ZIP-Providers	-1845188
25	(3 _Q :7 _{ZIP})	ZIP-Providers	-1849958
26	(9 _Q :1 _{ZIP})	Q-Providers	-1853561
27	(1 _Q :9 _{ZIP})	ZIP-Providers	-1860898
28	(4 _Q :6 _{ZIP})	ZIP-Providers	-1867223
29	(7 _Q :3 _{GD})	Q-Providers	-1877957
30	(6 _Q :4 _{GD})	ZIP-Providers	-1896519
31	(5 _Q :5 _{ZIP})	ZIP-Providers	-1913499
32	(8 _Q :2 _{ZIP})	Q-Providers	-1915929
33	(6 _Q :4 _{ZIP})	ZIP-Providers	-1964382
34	(5 _Q :5 _{GD})	Q-Providers	-1984321
35	(7 _Q :3 _{ZIP})	Q-Providers	-1996437

36	(7 _Q :3 _{GD})	ZIP-Providers	-2050831
37	(7 _Q :3 _{ZIP})	ZIP-Providers	-2051307
38	(6 _Q :4 _{ZIP})	Q-Providers	-2092973
39	(8 _Q :2 _{GD})	ZIP-Providers	-2139155
40	(8 _Q :2 _{ZIP})	ZIP-Providers	-2167170
41	(5 _Q :5 _{ZIP})	Q-Providers	-2252060
42	(1 _Q :9 _{ZIP})	GD-Providers	-2288641
43	(9 _Q :1 _{ZIP})	GD-Providers	-2298383
44	(1 _Q :9 _{GD})	GD-Providers	-2341907
45	(9 _Q :1 _{ZIP})	ZIP-Providers	-2346517
46	(6 _Q :4 _{ZIP})	GD-Providers	-2349538
47	(2 _Q :8 _{ZIP})	GD-Providers	-2359410
48	(8 _Q :2 _{ZIP})	GD-Providers	-2366778
49	(7 _Q :3 _{ZIP})	GD-Providers	-2368236
50	(5 _Q :5 _{ZIP})	GD-Providers	-2371040
51	(3 _Q :7 _{ZIP})	GD-Providers	-2375811
52	(4 _Q :6 _{ZIP})	GD-Providers	-2385268
53	(9 _Q :1 _{GD})	ZIP-Providers	-2422971
54	(4 _Q :6 _{ZIP})	Q-Providers	-2449528
55	(2 _Q :8 _{GD})	GD-Providers	-2596794
56	(8 _Q :2 _{GD})	Q-Providers	-2648372
57	(3 _Q :7 _{ZIP})	Q-Providers	-2675304
58	(1 _Q :9 _{ZIP})	Q-Providers	-2829670
59	(2 _Q :8 _{ZIP})	Q-Providers	-2830427
60	(4 _Q :6 _{GD})	GD-Providers	-3455369
61	(3 _Q :7 _{GD})	GD-Providers	-3497925
62	(5 _Q :5 _{GD})	GD-Providers	-4342554
63	(7 _Q :3 _{GD})	GD-Providers	-4366959
64	(6 _Q :4 _{GD})	GD-Providers	-4430651
65	(8 _Q :2 _{GD})	GD-Providers	-4802652
66	(9 _Q :1 _{GD})	GD-Providers	-9881069

Appendix C

MX/CS – Communication Protocol Specification

The following XML Schema represents the proof-of-concept realization of the developed message protocols in Section 5.3 on top of the *Job Submission and Definition Language*. The corresponding ontology file, created with Protégé (<<http://protege.stanford.edu>>, version 4.0.2), consists of an explicit formal definition of the concepts defined, their properties and relations to the well-known upper ontology *OpenCyc* (<<http://www.openecyc.org>>).

All the specifications in this thesis are open under the GNU Lesser General Public License as published by the Free Software Foundation, either as version 3 of the license, or any later version.

C.1 MX/CS – XML Schema Specification

```

1 <?xml version='1.0' encoding='UTF-8'?>
2 <xsd:schema xmlns:xsd='http://www.w3.org/2001/XMLSchema'
3   xmlns:bid='http://www.sormaproject.eu/message/ejsdl/beans'
4   xmlns:jsdl='http://schemas.ggf.org/jsdl/2005/11/jsdl'
5     xmlns:wsa='http://schemas.xmlsoap.org/ws/2004/03/addressing'
6     xmlns:sawsdl='http://www.w3.org/ns/sawsdl'
7     targetNamespace='http://www.sormaproject.eu/message/ejsdl/beans'
8     elementFormDefault='qualified'>
9
10  <xsd:import namespace='http://schemas.ggf.org/jsdl/2005/11/jsdl' schemaLocation='
11    jsdl.xsd' />
12  <xsd:import namespace='http://schemas.ggf.org/jsdl/2005/11/jsdl-posix'
13    schemaLocation='jsdl-posix.xsd' />
14
15  <!-- ===== -->
16  <!-- =====Economic Extensions ===== -->
17  <!-- ===== -->
18
19  <!-- ===== Private-Data Type ===== -->
20  <xsd:complexType name='PrivateData_Type'>
21    <xsd:sequence>
22      <xsd:element ref='bid:valuation' minOccurs='1' maxOccurs='1' />
23      <xsd:element ref='bid:strategy' minOccurs='0' maxOccurs='1' />
24      <xsd:element ref='bid:ScoringFunction' minOccurs='1' maxOccurs='1' />
25    </xsd:sequence>
26  </xsd:complexType>

```

```

25 <xsd:element name='PrivateData' type='bid:PrivateData_Type' sawsdl:modelReference=
    'mxcsProtocol#PrivateData' />
    <xsd:element name='valuation' type='xsd:double' sawsdl:modelReference='
        mxcsProtocol#valuation' />
    <xsd:element name='strategy' type='xsd:string' sawsdl:modelReference='mxcsProtocol
27 #strategy' />
    <xsd:element name='ScoringFunction' type='xsd:string' sawsdl:modelReference='
        mxcsProtocol#scoringFunction' />

29 <!-- ===== Public-Data Type ===== -->
    <xsd:complexType name='Bid_Type'>
31     <xsd:sequence>
33         <xsd:element ref='bid:bidPrice' minOccurs='0' maxOccurs='1' />
        <xsd:element ref='bid:signature' minOccurs='1' maxOccurs='1' />
        <xsd:element ref='bid:requestType' minOccurs='1' maxOccurs='1' />
35         <xsd:element ref='bid:participantId' minOccurs='1' maxOccurs='1' />
        <xsd:element ref='bid:expiration' minOccurs='0' maxOccurs='1' />
37         <xsd:element ref='bid:duration' minOccurs='0' maxOccurs='1' />
        <xsd:element ref='bid:serviceType' minOccurs='1' maxOccurs='1' />
39         <xsd:element ref='bid:paymentType' minOccurs='1' maxOccurs='1' />
        </xsd:sequence>
41     </xsd:complexType>

43 <!-- ===== Payment Type ===== -->
    <xsd:simpleType name='PaymentTypeEnumeration'>
45     <xsd:restriction base='xsd:string'>
        <xsd:enumeration value='BEFORE' />
47         <xsd:enumeration value='AFTER' />
        <xsd:enumeration value='EITHER' />
49     </xsd:restriction>
    </xsd:simpleType>

51 <!-- ===== Service Type ===== -->
53 <xsd:simpleType name='ServiceTypeEnumeration'>
    <xsd:restriction base='xsd:string'>
55     <xsd:enumeration value='WEBSERVICE' />
        <xsd:enumeration value='BATCH' />
57     </xsd:restriction>
    </xsd:simpleType>

59 <!-- ===== Request Type ===== -->
61 <xsd:simpleType name='RequestTypeEnumeration'>
    <xsd:restriction base='xsd:string'>
63     <xsd:enumeration value='BID' />
        <xsd:enumeration value='OFFER' />
65     <xsd:enumeration value='MATCH' />
    </xsd:restriction>
67 </xsd:simpleType>

69 <xsd:element name='paymentType' type='bid:PaymentTypeEnumeration'
    sawsdl:modelReference='mxcsProtocol#paymentType' />
    <xsd:element name='serviceType' type='bid:ServiceTypeEnumeration'
    sawsdl:modelReference='mxcsProtocol#serviceType' />
71 <xsd:element name='requestType' type='bid:RequestTypeEnumeration'
    sawsdl:modelReference='mxcsProtocol#requestType' />
    <xsd:element name='BidType' type='bid:Bid_Type' sawsdl:modelReference='
        mxcsProtocol#BidData' />
73 <xsd:element name='bidPrice' type='xsd:double' sawsdl:modelReference='mxcsProtocol
    #bid' />

```

```

75 <xsd:element name='signature' type='xsd:string' sawsdl:modelReference='
    mxcsProtocol#signature' />
77 <xsd:element name='participantId' type='xsd:string' sawsdl:modelReference='
    mxcsProtocol#participant' />
    <xsd:element name='expiration' type='xsd:long' sawsdl:modelReference='mxcsProtocol
    #expiration' />
77 <xsd:element name='duration' type='xsd:long' sawsdl:modelReference='mxcsProtocol#
    duration' />
79 <!-- ===== Market-Data Type ===== -->
81 <xsd:complexType name='MarketMessage_Type'>
    <xsd:sequence>
83     <xsd:element ref='bid:clearingPrice' minOccurs='1' maxOccurs='1' />
    <xsd:element ref='bid:contractId' minOccurs='1' maxOccurs='1' />
85     <xsd:element ref='bid:ConsumerContext' minOccurs='1' maxOccurs='1' />
    <xsd:element ref='bid:ProviderContext' minOccurs='1' maxOccurs='1' />
87     <xsd:element ref='bid:requestType' minOccurs='1' maxOccurs='1' />
    <xsd:element ref='bid:duration' minOccurs='0' maxOccurs='1' />
89     <xsd:element ref='bid:serviceType' minOccurs='1' maxOccurs='1' />
    <xsd:element ref='bid:paymentType' minOccurs='1' maxOccurs='1' />
91 </xsd:sequence>
    </xsd:complexType>
93 <xsd:element name='ConsumerContext' type='bid:Context_Type' sawsdl:modelReference='
    mxcsProtocol#ConsumerContext' />
95 <xsd:element name='ProviderContext' type='bid:Context_Type' sawsdl:modelReference='
    mxcsProtocol#ProviderContext' />
97 <xsd:complexType name='Context_Type'>
    <xsd:sequence>
99     <xsd:element ref='bid:signature' minOccurs='1' maxOccurs='1' />
    <xsd:element ref='bid:participantId' minOccurs='1' maxOccurs='1' />
101    <xsd:element ref='bid:bidId' minOccurs='1' maxOccurs='1' />
    </xsd:sequence>
103 </xsd:complexType>
105 <xsd:element name='MarketMessage' type='bid:MarketMessage_Type'
    sawsdl:modelReference='mxcsProtocol#MarketMessage' />
    <xsd:element name='clearingPrice' type='xsd:double' sawsdl:modelReference='
    mxcsProtocol#clearingPrice' />
107 <xsd:element name='contractId' type='xsd:string' sawsdl:modelReference='
    mxcsProtocol#contract' />
    <xsd:element name='consumerId' type='xsd:string' sawsdl:modelReference='
    mxcsProtocol#consumer' />
109 <xsd:element name='providerId' type='xsd:string' sawsdl:modelReference='
    mxcsProtocol#provider' />
    <xsd:element name='bidId' type='xsd:string' sawsdl:modelReference='mxcsProtocol#id
    ' />
111 <!-- ===== Penalty Type ===== -->
113 <xsd:complexType name='Penalty_Type'>
    <xsd:sequence>
115     <xsd:element ref='bid:functionName' minOccurs='1' maxOccurs='1' />
    <xsd:element ref='bid:normalizationConstant' minOccurs='1' maxOccurs='1' />
117 </xsd:sequence>
    </xsd:complexType>
119 <xsd:element name='Penalty' type='bid:Penalty_Type' sawsdl:modelReference='http:
    //www.im.uni-karlsruhe.de/sorma/fileadmin/ontology/mxcsProtocol#Penalty' />

```



```

121 <xsd:element name='functionName' type='bid:FunctionNameEnumeration'
    sawsdl:modelReference='http://www.im.uni-karlsruhe.de/sorma/fileadmin/ontology
    /mxcsProtocol#Function' />
123 <xsd:element name='normalizationConstant' type='xsd:double' sawsdl:modelReference=
    'http://www.im.uni-karlsruhe.de/sorma/fileadmin/ontology/mxcsProtocol#
    NormalizationFactor' />
125 <xsd:simpleType name='FunctionNameEnumeration'>
    <xsd:restriction base='xsd:string'>
127 <xsd:enumeration value='DefaultPenalty' />
    </xsd:restriction>
    </xsd:simpleType>
129 <!-- =====>
131 <!-- ===== Private Message =====>
133 <xsd:complexType name='PrivateDefinition.Type'>
    <xsd:sequence>
135 <xsd:element ref='jsdl:JobDefinition' minOccurs='0' maxOccurs='1' />
    <xsd:element ref='bid:PrivateData' minOccurs='1' maxOccurs='1' />
137 <xsd:element ref='bid:Bid' minOccurs='1' maxOccurs='1' />
    <xsd:element ref='bid:Penalty' minOccurs='0' maxOccurs='1' />
    </xsd:sequence>
139 <xsd:attribute name='requestId' type='xsd:string' use='required'
    sawsdl:modelReference='mxcsProtocol#requestId' />
    </xsd:complexType>
141 <xsd:element name='PrivateDefinition' type='bid:PrivateDefinition.Type' />
143 <!-- =====>
145 <!-- ===== PublicMessage =====>
147 <xsd:complexType name='BidDefinition.Type'>
    <xsd:sequence>
149 <xsd:element ref='jsdl:JobDefinition' minOccurs='0' maxOccurs='1' />
    <xsd:element ref='bid:Bid' minOccurs='1' maxOccurs='1' />
    <xsd:element ref='bid:Penalty' minOccurs='0' maxOccurs='1' />
    </xsd:sequence>
151 <xsd:attribute name='bidId' type='xsd:string' use='required'
    sawsdl:modelReference='mxcsProtocol#id' />
153 </xsd:complexType>
155 <xsd:element name='BidDefinition' type='bid:BidDefinition.Type' />
157 <!-- =====>
159 <!-- ===== MarketMessage =====>
161 <xsd:complexType name='MarketDefinition.Type'>
    <xsd:sequence>
163 <xsd:element ref='jsdl:JobDefinition' minOccurs='0' maxOccurs='1' />
    <xsd:element ref='bid:MarketMessage' minOccurs='1' maxOccurs='1' />
165 <xsd:element ref='bid:Penalty' minOccurs='0' maxOccurs='1' />
    </xsd:sequence>
167 <xsd:attribute name='marketMessageId' type='xsd:string' use='required'
    sawsdl:modelReference='mxcsProtocol#contract' />
    </xsd:complexType>
169 <xsd:element name='MarketDefinition' type='bid:MarketDefinition.Type'
    sawsdl:modelReference='mxcsProtocol#MarketData' />
171

```

```
</xsd:schema>
```

C.2 MX/CS – OWL Specification

```

2 <?xml version="1.0"?>
3 <!DOCTYPE Ontology [
4   <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
5   <!ENTITY dc "http://purl.org/dc/elements/1.1/" >
6   <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
7   <!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#" >
8   <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
9   <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
10  <!ENTITY mxcsProtocol "http://www.im.uni-karlsruhe.de/sorma/fileadmin/ontology/
11    mxcsProtocol.owl#" >
12 ]>
13 <Ontology xmlns="http://www.w3.org/2006/12/owl2-xml#"
14   xml:base="http://www.w3.org/2006/12/owl2-xml#"
15   xmlns:dc="http://purl.org/dc/elements/1.1/"
16   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
17   xmlns:mxcsProtocol="http://www.im.uni-karlsruhe.de/sorma/fileadmin/ontology/
18     mxcsProtocol.owl#"
19   xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
20   xmlns:owl="http://www.w3.org/2002/07/owl#"
21   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
22   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
23   URI="http://www.im.uni-karlsruhe.de/sorma/fileadmin/ontology/mxcsProtocol.owl">
24   <EntityAnnotation>
25     <Class URI="\&mxcsProtocol;MarketInformation" />
26     <Annotation annotationURI="\&dc;relation">
27       <Constant>http://sw.opencyc.org/concept/Mx4raTUz-O-KEdyAAAACs6hbjw</
28       Constant>
29     </Annotation>
30   </EntityAnnotation>
31   <EntityAnnotation>
32     <Class URI="\&mxcsProtocol;MarketInformation" />
33     <Annotation annotationURI="\&rdfs;comment">
34       <Constant>Market information contains data of the quantity, type, bids
35       and clearing prices of the traded goods or services.</Constant>
36     </Annotation>
37   </EntityAnnotation>
38   <Declaration>
39     <Class URI="\&mxcsProtocol;MarketInformation" />
40   </Declaration>
41   <SubClassOf>
42     <Class URI="\&mxcsProtocol;MarketMessage" />
43     <Class URI="\&owl;Thing" />
44   </SubClassOf>
45   <EntityAnnotation>
46     <Class URI="\&mxcsProtocol;MarketMessage" />
47     <Annotation annotationURI="\&rdfs;comment">
48       <Constant>The context of a market document containing data, which is
49       generated by the market mechanism and submitted to the target
50       consumer and provider as well as related parties e.g. for contract
51       and service level agreement management.</Constant>
52     </Annotation>
53   </EntityAnnotation>
54   <EntityAnnotation>

```

```

48     <Class URI="\&mxcsProtocol;MarketMessage" />
    <Annotation annotationURI="\&rdfs;seeAlso">
        <Constant>http://sw.opencyc.org/concept/Mx4rvViA1ZwpEbGdrcN5Y29ycA</
50         Constant>
    </Annotation>
</EntityAnnotation>
52 <Declaration>
    <Class URI="\&mxcsProtocol;MarketMessage" />
54 </Declaration>
<SubClassOf>
56     <Class URI="\&mxcsProtocol;PrivateMessage" />
    <Class URI="\&owl;Thing" />
58 </SubClassOf>
<EntityAnnotation>
60     <Class URI="\&mxcsProtocol;PrivateMessage" />
    <Annotation annotationURI="\&rdfs;comment">
62         <Constant>The context of a consumer or provider document, that contains
            private economic data, available only to the bidding agent of the
            consumer or provider.
The bidding agent is a local program to the consumer or provider, which utilize a
selected bidding strategy and based on the economic data, it generates bids,
which are submitted to the market.</Constant>
64     </Annotation>
</EntityAnnotation>
66 <Declaration>
    <Class URI="\&mxcsProtocol;PrivateMessage" />
68 </Declaration>
<SubClassOf>
70     <Class URI="\&mxcsProtocol;PublicMessage" />
    <Class URI="\&owl;Thing" />
72 </SubClassOf>
<EntityAnnotation>
74     <Class URI="\&mxcsProtocol;PublicMessage" />
    <Annotation annotationURI="\&rdfs;comment">
76         <Constant>The context of a consumer or provider document, that contains
            public economic data, submitted to the target market mechanism.</
            Constant>
    </Annotation>
78 </EntityAnnotation>
<Declaration>
80     <Class URI="\&mxcsProtocol;PublicMessage" />
</Declaration>
82 <EntityAnnotation>
    <Class URI="\&mxcsProtocol;StateMessage" />
84     <Annotation annotationURI="\&dc;relation">
        <Constant>http://sw.opencyc.org/concept/Mx4rZqZ6chY-Ed2AAAACs6hRXg</
            Constant>
86     </Annotation>
</EntityAnnotation>
88 <EntityAnnotation>
    <Class URI="\&mxcsProtocol;StateMessage" />
90     <Annotation annotationURI="\&rdfs;comment">
        <Constant>Status represents an actual monitoring data or fact
            information about a good, trading object, service, job or
            application artifact, which is valid for the time of request, but
            can change in the time.</Constant>
92     </Annotation>
</EntityAnnotation>
94 <Declaration>
    <Class URI="\&mxcsProtocol;StateMessage" />

```

```

96     </Declaration>
97     <DataPropertyDomain>
98         <DataProperty URI="\&mxcsProtocol;auction" />
99         <Class URI="\&mxcsProtocol;MarketInformation" />
100     </DataPropertyDomain>
101     <DataPropertyRange>
102         <DataProperty URI="\&mxcsProtocol;auction" />
103         <Datatype URI="\&xsd:string" />
104     </DataPropertyRange>
105     <EntityAnnotation>
106         <DataProperty URI="\&mxcsProtocol;auction" />
107         <Annotation annotationURI="\&dc:relation">
108             <Constant>http://sw.opencyc.org/concept/Mx4rvotkAJwpEbGdrcN5Y29ycA</
109                 Constant>
110         </Annotation>
111     </EntityAnnotation>
112     <EntityAnnotation>
113         <DataProperty URI="\&mxcsProtocol;auction" />
114         <Annotation annotationURI="\&rdfs:comment">
115             <Constant>An auction is a process of buying and selling goods or
116                 services by offering them up for bid, taking bids, and then selling
117                 the item to the highest bidder. In economic theory, an auction may
118                 refer to any mechanism or set of trading rules for exchange.
119             [http://en.wikipedia.org/wiki/Auction]</Constant>
120         </Annotation>
121     </EntityAnnotation>
122     <Declaration>
123         <DataProperty URI="\&mxcsProtocol;auction" />
124     </Declaration>
125     <DataPropertyDomain>
126         <DataProperty URI="\&mxcsProtocol;bid" />
127         <Class URI="\&mxcsProtocol;MarketInformation" />
128     </DataPropertyDomain>
129     <DataPropertyDomain>
130         <DataProperty URI="\&mxcsProtocol;bid" />
131         <Class URI="\&mxcsProtocol;PublicMessage" />
132     </DataPropertyDomain>
133     <DataPropertyRange>
134         <DataProperty URI="\&mxcsProtocol;bid" />
135         <Datatype URI="\&xsd:double" />
136     </DataPropertyRange>
137     <EntityAnnotation>
138         <DataProperty URI="\&mxcsProtocol;bid" />
139         <Annotation annotationURI="\&dc:relation">
140             <Constant>http://sw.opencyc.org/concept/Mx4rcLZ9AfHEduAAADggVaqv</
141                 Constant>
142         </Annotation>
143     </EntityAnnotation>
144     <EntityAnnotation>
145         <DataProperty URI="\&mxcsProtocol;bid" />
146         <Annotation annotationURI="\&rdfs:comment">
147             <Constant>The generated bid price, based on the selected bidding
148                 strategy, of the consumer or provider. </Constant>
149         </Annotation>
150     </EntityAnnotation>
151     <Declaration>
152         <DataProperty URI="\&mxcsProtocol;bid" />
153     </Declaration>
154     <SubDataPropertyOf>
155         <DataProperty URI="\&mxcsProtocol;bidId" />

```

```

150     <DataProperty URI="\&mxcsProtocol;consumerContext" />
151 </SubDataPropertyOf>
152 <SubDataPropertyOf>
153     <DataProperty URI="\&mxcsProtocol;bidId" />
154     <DataProperty URI="\&mxcsProtocol;providerContext" />
155 </SubDataPropertyOf>
156 <DataPropertyDomain>
157     <DataProperty URI="\&mxcsProtocol;bidId" />
158     <Class URI="\&mxcsProtocol;PublicMessage" />
159 </DataPropertyDomain>
160 <DataPropertyRange>
161     <DataProperty URI="\&mxcsProtocol;bidId" />
162     <Datatype URI="\&xsd;anyType" />
163 </DataPropertyRange>
164 <EntityAnnotation>
165     <DataProperty URI="\&mxcsProtocol;bidId" />
166     <Annotation annotationURI="\&dc;relation">
167         <Constant>http://sw.opencyc.org/concept/Mx4rvVi7GJwpEbGdrcN5Y29ycA</
168             Constant>
169     </Annotation>
170 </EntityAnnotation>
171 <EntityAnnotation>
172     <DataProperty URI="\&mxcsProtocol;bidId" />
173     <Annotation annotationURI="\&rdfs;comment">
174         <Constant>Unique identifacation of a bid in form of a number or string.<
175             /Constant>
176     </Annotation>
177 </EntityAnnotation>
178 <Declaration>
179     <DataProperty URI="\&mxcsProtocol;bidId" />
180 </Declaration>
181 <DataPropertyDomain>
182     <DataProperty URI="\&mxcsProtocol;clearingPrice" />
183     <Class URI="\&mxcsProtocol;MarketMessage" />
184 </DataPropertyDomain>
185 <DataPropertyRange>
186     <DataProperty URI="\&mxcsProtocol;clearingPrice" />
187     <Datatype URI="\&xsd;double" />
188 </DataPropertyRange>
189 <EntityAnnotation>
190     <DataProperty URI="\&mxcsProtocol;clearingPrice" />
191     <Annotation annotationURI="\&dc;relation">
192         <Constant>http://sw.opencyc.org/concept/Mx4rvViMf5wpEbGdrcN5Y29ycA</
193             Constant>
194     </Annotation>
195 </EntityAnnotation>
196 <EntityAnnotation>
197     <DataProperty URI="\&mxcsProtocol;clearingPrice" />
198     <Annotation annotationURI="\&dc;relation">
199         <Constant>http://sw.opencyc.org/concept/Mx4rvbYa8JwpEbGdrcN5Y29ycA</
200             Constant>
201     </Annotation>
202 </EntityAnnotation>
203 <EntityAnnotation>
204     <DataProperty URI="\&mxcsProtocol;clearingPrice" />
205     <Annotation annotationURI="\&rdfs;comment">
206         <Constant>The specified monetary value assigned to an artifact on market
207             clearing , called also market-clearing price. In an auction-based
208             scenario, the price is determined by the auction, based on the bids
209             and offers of providers and consumers interested in trading that

```

```

                artifact. The price that a consumer has to pay to a provider in
                order to execute his job on the target provider machine.</Constant>
            </Annotation>
204 </EntityAnnotation>
        <EntityAnnotation>
206         <DataProperty URI="\&mxcsProtocol;clearingPrice" />
            <Annotation annotationURI="\&rdfs;isDefinedBy">
208             <Constant>http://www-personal.umich.edu/~alandear/glossary/m.html</
                Constant>
            </Annotation>
210 </EntityAnnotation>
        <Declaration>
212         <DataProperty URI="\&mxcsProtocol;clearingPrice" />
        </Declaration>
214 <DataPropertyDomain>
            <DataProperty URI="\&mxcsProtocol;completionTime" />
216         <Class URI="\&mxcsProtocol;StateMessage" />
        </DataPropertyDomain>
218 <DataPropertyRange>
            <DataProperty URI="\&mxcsProtocol;completionTime" />
220         <Datatype URI="\&xsd;double" />
        </DataPropertyRange>
222 <EntityAnnotation>
            <DataProperty URI="\&mxcsProtocol;completionTime" />
224         <Annotation annotationURI="\&dc;relation">
            <Constant>http://sw.opencyc.org/concept/Mx4rvVjXppwpEbGdrcN5Y29ycA</
                Constant>
226         </Annotation>
        </EntityAnnotation>
228 <EntityAnnotation>
            <DataProperty URI="\&mxcsProtocol;completionTime" />
230         <Annotation annotationURI="\&rdfs;comment">
            <Constant>The total time needed to complete a job, application or
                service execution.</Constant>
232         </Annotation>
        </EntityAnnotation>
234 <Declaration>
            <DataProperty URI="\&mxcsProtocol;completionTime" />
236 </Declaration>
        <DataPropertyDomain>
238         <DataProperty URI="\&mxcsProtocol;consumer" />
            <Class URI="\&mxcsProtocol;MarketMessage" />
240 </DataPropertyDomain>
        <DataPropertyRange>
242         <DataProperty URI="\&mxcsProtocol;consumer" />
            <Datatype URI="\&xsd;anyType" />
244 </DataPropertyRange>
        <EntityAnnotation>
246         <DataProperty URI="\&mxcsProtocol;consumer" />
            <Annotation annotationURI="\&dc;relation">
248             <Constant>http://sw.opencyc.org/concept/Mx4rvVjK1JwpEbGdrcN5Y29ycA</
                Constant>
            </Annotation>
250 </EntityAnnotation>
        <EntityAnnotation>
252         <DataProperty URI="\&mxcsProtocol;consumer" />
            <Annotation annotationURI="\&rdfs;comment">
254             <Constant>A consumer (costumer) has an intention to purchase goods or
                services. A consumer might make the purchase either directly or
                through a bidding agent. </Constant>

```

```

256     </Annotation>
257 </EntityAnnotation>
258 <Declaration>
259   <DataProperty URI="\&mxcsProtocol;consumer" />
260 </Declaration>
261 <DataPropertyDomain>
262   <DataProperty URI="\&mxcsProtocol;consumerContext" />
263   <Class URI="\&mxcsProtocol;MarketMessage" />
264 </DataPropertyDomain>
265 <EntityAnnotation>
266   <DataProperty URI="\&mxcsProtocol;consumerContext" />
267   <Annotation annotationURI="&dc;relation">
268     <Constant>http://sw.opencyc.org/concept/Mx4rvViA1ZwpEbGdrcN5Y29ycA</
269       Constant>
270   </Annotation>
271 </EntityAnnotation>
272 <EntityAnnotation>
273   <DataProperty URI="\&mxcsProtocol;consumerContext" />
274   <Annotation annotationURI="&rdfs;comment">
275     <Constant>This property groups a set of common consumer properties.</
276       Constant>
277   </Annotation>
278 </EntityAnnotation>
279 <Declaration>
280   <DataProperty URI="\&mxcsProtocol;consumerContext" />
281 </Declaration>
282 <DataPropertyDomain>
283   <DataProperty URI="\&mxcsProtocol;contract" />
284   <Class URI="\&mxcsProtocol;MarketMessage" />
285 </DataPropertyDomain>
286 <DataPropertyRange>
287   <DataProperty URI="\&mxcsProtocol;contract" />
288   <Datatype URI="&xsd;anyType" />
289 </DataPropertyRange>
290 <EntityAnnotation>
291   <DataProperty URI="\&mxcsProtocol;contract" />
292   <Annotation annotationURI="&dc;relation">
293     <Constant>http://sw.opencyc.org/concept/Mx4rvZvacpwpEbGdrcN5Y29ycA</
294       Constant>
295   </Annotation>
296 </EntityAnnotation>
297 <EntityAnnotation>
298   <DataProperty URI="\&mxcsProtocol;contract" />
299   <Annotation annotationURI="&rdfs;comment">
300     <Constant>A collection of agreements. Each instance is a legal agreement
301       in which two or more agreeing agents promise to do (or not do)
302       something. There are legal consequences to breaking the promises
303       made in a contract.</Constant>
304   </Annotation>
305 </EntityAnnotation>
306 <Declaration>
307   <DataProperty URI="\&mxcsProtocol;contract" />
308 </Declaration>
309 <DataPropertyDomain>
310   <DataProperty URI="\&mxcsProtocol;duration" />
311   <Class URI="\&mxcsProtocol;MarketMessage" />
312 </DataPropertyDomain>
313 <DataPropertyDomain>
314   <DataProperty URI="\&mxcsProtocol;duration" />
315   <Class URI="\&mxcsProtocol;PublicMessage" />

```

```

310 </DataPropertyDomain>
311 <DataPropertyDomain>
312   <DataProperty URI="\&mxcsProtocol;duration" />
313   <Class URI="\&mxcsProtocol;StateMessage" />
314 </DataPropertyDomain>
315 <DataPropertyRange>
316   <DataProperty URI="\&mxcsProtocol;duration" />
317   <Datatype URI="\&xsd;long" />
318 </DataPropertyRange>
319 <EntityAnnotation>
320   <DataProperty URI="\&mxcsProtocol;duration" />
321   <Annotation annotationURI="\&dc;relation">
322     <Constant>http://sw.opencyc.org/concept/Mx4rvVijs5wpEbGdrcN5Y29ycA</
323     Constant>
324   </Annotation>
325 </EntityAnnotation>
326 <EntityAnnotation>
327   <DataProperty URI="\&mxcsProtocol;duration" />
328   <Annotation annotationURI="\&rdfs;comment">
329     <Constant>A measurable quantity that relates a temporal thing to the
330     length of time, in milliseconds, during which it existed, happened,
331     or obtained. </Constant>
332   </Annotation>
333 </EntityAnnotation>
334 <Declaration>
335   <DataProperty URI="\&mxcsProtocol;duration" />
336 </Declaration>
337 <DataPropertyDomain>
338   <DataProperty URI="\&mxcsProtocol;finalPrice" />
339   <Class URI="\&mxcsProtocol;StateMessage" />
340 </DataPropertyDomain>
341 <DataPropertyRange>
342   <DataProperty URI="\&mxcsProtocol;finalPrice" />
343   <Datatype URI="\&xsd;double" />
344 </DataPropertyRange>
345 <EntityAnnotation>
346   <DataProperty URI="\&mxcsProtocol;finalPrice" />
347   <Annotation annotationURI="\&dc;relation">
348     <Constant>http://sw.opencyc.org/concept/Mx4rvViMf5wpEbGdrcN5Y29ycA</
349     Constant>
350   </Annotation>
351 </EntityAnnotation>
352 <EntityAnnotation>
353   <DataProperty URI="\&mxcsProtocol;finalPrice" />
354   <Annotation annotationURI="\&rdfs;comment">
355     <Constant>The final price is the market clearing price after considering
356     the penalties.</Constant>
357   </Annotation>
358 </EntityAnnotation>
359 <Declaration>
360   <DataProperty URI="\&mxcsProtocol;finalPrice" />
361 </Declaration>
362 <DataPropertyDomain>
363   <DataProperty URI="\&mxcsProtocol;parameter" />
364   <Class URI="\&mxcsProtocol;StateMessage" />
365 </DataPropertyDomain>
366 <DataPropertyRange>
367   <DataProperty URI="\&mxcsProtocol;parameter" />
368   <Datatype URI="\&xsd;anyType" />
369 </DataPropertyRange>

```



```

364 <EntityAnnotation>
      <DataProperty URI="\&mxcsProtocol;parameter" />
366 <Annotation annotationURI="\&dc;relation">
      <Constant>Value or collection of parameters and their values.</Constant>
368 </Annotation>
</EntityAnnotation>
370 <EntityAnnotation>
      <DataProperty URI="\&mxcsProtocol;parameter" />
372 <Annotation annotationURI="\&dc;relation">
      <Constant>http://sw.opencyc.org/concept/Mx4r9IvlpcoEdqAAAACs4vPlg</
      Constant>
374 </Annotation>
</EntityAnnotation>
376 <Declaration>
      <DataProperty URI="\&mxcsProtocol;parameter" />
378 </Declaration>
<SubDataPropertyOf>
380 <DataProperty URI="\&mxcsProtocol;participant" />
      <DataProperty URI="\&mxcsProtocol;consumerContext" />
382 </SubDataPropertyOf>
<SubDataPropertyOf>
384 <DataProperty URI="\&mxcsProtocol;participant" />
      <DataProperty URI="\&mxcsProtocol;providerContext" />
386 </SubDataPropertyOf>
<DataPropertyDomain>
388 <DataProperty URI="\&mxcsProtocol;participant" />
      <Class URI="\&mxcsProtocol;PublicMessage" />
390 </DataPropertyDomain>
<DataPropertyRange>
392 <DataProperty URI="\&mxcsProtocol;participant" />
      <Datatype URI="\&xsd;anyType" />
394 </DataPropertyRange>
<EntityAnnotation>
396 <DataProperty URI="\&mxcsProtocol;participant" />
      <Annotation annotationURI="\&rdfs;comment">
398 <Constant>An identifier of the consumer or provider actor. </Constant>
      </Annotation>
400 </EntityAnnotation>
<EntityAnnotation>
402 <DataProperty URI="\&mxcsProtocol;participant" />
      <Annotation annotationURI="\&rdfs;comment">
404 <Constant>Participant is a consumer, provider, consumer agent, provider
      agent, auctioneer or trading manager.</Constant>
      </Annotation>
406 </EntityAnnotation>
<Declaration>
408 <DataProperty URI="\&mxcsProtocol;participant" />
</Declaration>
410 <DataPropertyDomain>
      <DataProperty URI="\&mxcsProtocol;paymentType" />
412 <Class URI="\&mxcsProtocol;MarketMessage" />
</DataPropertyDomain>
414 <DataPropertyDomain>
      <DataProperty URI="\&mxcsProtocol;paymentType" />
416 <Class URI="\&mxcsProtocol;PublicMessage" />
</DataPropertyDomain>
418 <DataPropertyRange>
      <DataProperty URI="\&mxcsProtocol;paymentType" />
420 <Datatype URI="\&xsd;string" />
</DataPropertyRange>

```

```

422     <EntityAnnotation>
423         <DataProperty URI="\&mxcsProtocol;paymentType" />
424         <Annotation annotationURI="\&dc;relation">
425             <Constant>http://sw.opencyc.org/concept/Mx4rvVkKHZwpEbGdrcN5Y29ycA</
426                 Constant>
427         </Annotation>
428     </EntityAnnotation>
429     <EntityAnnotation>
430         <DataProperty URI="\&mxcsProtocol;paymentType" />
431         <Annotation annotationURI="\&rdfs;comment">
432             <Constant>The type of payment, BEFORE or AFTER job execution.</Constant>
433         </Annotation>
434     </EntityAnnotation>
435     <Declaration>
436         <DataProperty URI="\&mxcsProtocol;paymentType" />
437     </Declaration>
438     <DataPropertyDomain>
439         <DataProperty URI="\&mxcsProtocol;provider" />
440         <Class URI="\&mxcsProtocol;MarketMessage" />
441     </DataPropertyDomain>
442     <DataPropertyRange>
443         <DataProperty URI="\&mxcsProtocol;provider" />
444         <Datatype URI="\&xsd;anyType" />
445     </DataPropertyRange>
446     <EntityAnnotation>
447         <DataProperty URI="\&mxcsProtocol;provider" />
448         <Annotation annotationURI="\&dc;relation">
449             <Constant>http://sw.opencyc.org/concept/Mx4rvViQGpwpEbGdrcN5Y29ycA</
450                 Constant>
451         </Annotation>
452     </EntityAnnotation>
453     <EntityAnnotation>
454         <DataProperty URI="\&mxcsProtocol;provider" />
455         <Annotation annotationURI="\&rdfs;comment">
456             <Constant>A provider (seller) has an intention to offer (sell) goods or
457                 services. A provider might provide its good or services either
458                 directly or through a bidding agent. </Constant>
459         </Annotation>
460     </EntityAnnotation>
461     <Declaration>
462         <DataProperty URI="\&mxcsProtocol;provider" />
463     </Declaration>
464     <DataPropertyDomain>
465         <DataProperty URI="\&mxcsProtocol;providerContext" />
466         <Class URI="\&mxcsProtocol;MarketMessage" />
467     </DataPropertyDomain>
468     <EntityAnnotation>
469         <DataProperty URI="\&mxcsProtocol;providerContext" />
470         <Annotation annotationURI="\&dc;relation">
471             <Constant>http://sw.opencyc.org/concept/Mx4rvViA1ZwpEbGdrcN5Y29ycA</
472                 Constant>
473         </Annotation>
474     </EntityAnnotation>
475     <EntityAnnotation>
476         <DataProperty URI="\&mxcsProtocol;providerContext" />
477         <Annotation annotationURI="\&rdfs;comment">
478             <Constant>This property groups a set of common provider properties.</
479                 Constant>
480         </Annotation>
481     </EntityAnnotation>

```

```

476 <Declaration>
477   <DataProperty URI="\&mxcsProtocol;providerContext" />
478 </Declaration>
479 <DataPropertyDomain>
480   <DataProperty URI="\&mxcsProtocol;query" />
481   <Class URI="\&mxcsProtocol;MarketInformation" />
482 </DataPropertyDomain>
483 <DataPropertyRange>
484   <DataProperty URI="\&mxcsProtocol;query" />
485   <Datatype URI="\&xsd;anyType" />
486 </DataPropertyRange>
487 <EntityAnnotation>
488   <DataProperty URI="\&mxcsProtocol;query" />
489   <Annotation annotationURI="\&dc;relation">
490     <Constant>Request of a specific information about a specific artifact ,
491       auction , object , trading object and market participant.</Constant>
492   </Annotation>
493 </EntityAnnotation>
494 <EntityAnnotation>
495   <DataProperty URI="\&mxcsProtocol;query" />
496   <Annotation annotationURI="\&dc;relation">
497     <Constant>http://sw.opencyc.org/concept/Mx4rvVitwpwpEbGdrcN5Y29ycA</
498     Constant>
499   </Annotation>
500 </EntityAnnotation>
501 <Declaration>
502   <DataProperty URI="\&mxcsProtocol;query" />
503 </Declaration>
504 <DataPropertyDomain>
505   <DataProperty URI="\&mxcsProtocol;requestType" />
506   <Class URI="\&mxcsProtocol;MarketMessage" />
507 </DataPropertyDomain>
508 <DataPropertyDomain>
509   <DataProperty URI="\&mxcsProtocol;requestType" />
510   <Class URI="\&mxcsProtocol;PublicMessage" />
511 </DataPropertyDomain>
512 <DataPropertyRange>
513   <DataProperty URI="\&mxcsProtocol;requestType" />
514   <Datatype URI="\&xsd;string" />
515 </DataPropertyRange>
516 <EntityAnnotation>
517   <DataProperty URI="\&mxcsProtocol;requestType" />
518   <Annotation annotationURI="\&dc;relation">
519     <Constant>http://sw.opencyc.org/concept/Mx4rvVrH55wpEbGdrcN5Y29ycA</
520     Constant>
521   </Annotation>
522 </EntityAnnotation>
523 <EntityAnnotation>
524   <DataProperty URI="\&mxcsProtocol;requestType" />
525   <Annotation annotationURI="\&rdfs;comment">
526     <Constant>The type of the request e.g. BID, OFFER, MATCH.</Constant>
527   </Annotation>
528 </EntityAnnotation>
529 <Declaration>
530   <DataProperty URI="\&mxcsProtocol;requestType" />
531 </Declaration>
532 <DataPropertyDomain>
533   <DataProperty URI="\&mxcsProtocol;response" />
534   <Class URI="\&mxcsProtocol;MarketInformation" />
535 </DataPropertyDomain>

```

```

534     <DataPropertyRange>
535         <DataProperty URI="\&mxcsProtocol;response" />
536         <Datatype URI="\&xsd:anyType" />
537     </DataPropertyRange>
538     <EntityAnnotation>
539         <DataProperty URI="\&mxcsProtocol;response" />
540         <Annotation annotationURI="\&dc:relation">
541             <Constant>http://sw.opencyc.org/concept/Mx4rvViyOJwpEbGdrcN5Y29ycA</
542                 Constant>
543         </Annotation>
544     </EntityAnnotation>
545     <EntityAnnotation>
546         <DataProperty URI="\&mxcsProtocol;response" />
547         <Annotation annotationURI="\&rdfs:comment">
548             <Constant>Providing a information to an object or agent on a method
549                 invocaton.</Constant>
550         </Annotation>
551     </EntityAnnotation>
552     <Declaration>
553         <DataProperty URI="\&mxcsProtocol;response" />
554     </Declaration>
555     <DataPropertyDomain>
556         <DataProperty URI="\&mxcsProtocol;scoringFunction" />
557         <Class URI="\&mxcsProtocol;StateMessage" />
558     </DataPropertyDomain>
559     <DataPropertyRange>
560         <DataProperty URI="\&mxcsProtocol;scoringFunction" />
561         <Datatype URI="\&xsd:string" />
562     </DataPropertyRange>
563     <EntityAnnotation>
564         <DataProperty URI="\&mxcsProtocol;scoringFunction" />
565         <Annotation annotationURI="\&dc:relation">
566             <Constant>http://sw.opencyc.org/concept/Mx4rvVi9LZwpEbGdrcN5Y29ycA</
567                 Constant>
568         </Annotation>
569     </EntityAnnotation>
570     <EntityAnnotation>
571         <DataProperty URI="\&mxcsProtocol;scoringFunction" />
572         <Annotation annotationURI="\&rdfs:comment">
573             <Constant>A specification of a goal, based on well-defined and
574                 measurable criterias.</Constant>
575         </Annotation>
576     </EntityAnnotation>
577     <Declaration>
578         <DataProperty URI="\&mxcsProtocol;scoringFunction" />
579     </Declaration>
580     <DataPropertyDomain>
581         <DataProperty URI="\&mxcsProtocol;serviceType" />
582         <Class URI="\&mxcsProtocol;MarketMessage" />
583     </DataPropertyDomain>
584     <DataPropertyDomain>
585         <DataProperty URI="\&mxcsProtocol;serviceType" />
586         <Class URI="\&mxcsProtocol;PublicMessage" />
587     </DataPropertyDomain>
588     <DataPropertyRange>
589         <DataProperty URI="\&mxcsProtocol;serviceType" />
590         <Datatype URI="\&xsd:string" />
591     </DataPropertyRange>
592     <EntityAnnotation>
593         <DataProperty URI="\&mxcsProtocol;serviceType" />

```

```

590     <Annotation annotationURI="&dc;relation">
591         <Constant>http://sw.opencyc.org/concept/Mx4rkFbJUKPjQdmfp4wat978Fw</
592         Constant>
593     </Annotation>
594 </EntityAnnotation>
595 <EntityAnnotation>
596     <DataProperty URI="\&mxcsProtocol;serviceType" />
597     <Annotation annotationURI="&rdfs;comment">
598         <Constant>The type of the requested service e.g. command line batch job
599         execution (BATCH) or web application (WEBSERVICE).</Constant>
600     </Annotation>
601 </EntityAnnotation>
602 <Declaration>
603     <DataProperty URI="\&mxcsProtocol;serviceType" />
604 </Declaration>
605 <DataPropertyDomain>
606     <DataProperty URI="\&mxcsProtocol;signature" />
607     <Class URI="\&mxcsProtocol;MarketMessage" />
608 </DataPropertyDomain>
609 <DataPropertyDomain>
610     <DataProperty URI="\&mxcsProtocol;signature" />
611     <Class URI="\&mxcsProtocol;PublicMessage" />
612 </DataPropertyDomain>
613 <DataPropertyRange>
614     <DataProperty URI="\&mxcsProtocol;signature" />
615     <Datatype URI="&xsd;anyType" />
616 </DataPropertyRange>
617 <EntityAnnotation>
618     <DataProperty URI="\&mxcsProtocol;signature" />
619     <Annotation annotationURI="&dc;relation">
620         <Constant>http://sw.opencyc.org/concept/Mx4rvVjNKpwpEbGdrcN5Y29ycA</
621         Constant>
622     </Annotation>
623 </EntityAnnotation>
624 <EntityAnnotation>
625     <DataProperty URI="\&mxcsProtocol;signature" />
626     <Annotation annotationURI="&rdfs;comment">
627         <Constant>A digital sequence, which represents a valid and legal &#39;
628         fingermark&#39; of a trading party, market participant, consumer,
629         provider or auctioneer.</Constant>
630     </Annotation>
631 </EntityAnnotation>
632 <Declaration>
633     <DataProperty URI="\&mxcsProtocol;signature" />
634 </Declaration>
635 <DataPropertyDomain>
636     <DataProperty URI="\&mxcsProtocol;strategy" />
637     <Class URI="\&mxcsProtocol;PrivateMessage" />
638 </DataPropertyDomain>
639 <DataPropertyRange>
640     <DataProperty URI="\&mxcsProtocol;strategy" />
641     <Datatype URI="&xsd;string" />
642 </DataPropertyRange>
643 <EntityAnnotation>
644     <DataProperty URI="\&mxcsProtocol;strategy" />
645     <Annotation annotationURI="&dc;relation">
646         <Constant>http://sw.opencyc.org/concept/Mx4rvViB_5wpEbGdrcN5Y29ycA</
647         Constant>
648     </Annotation>
649 </EntityAnnotation>

```

```

644     <EntityAnnotation>
        <DataProperty URI="\&mxcsProtocol;strategy" />
646     <Annotation annotationURI="\&rdfs;comment">
        <Constant>A bidding strategy is a complete plan of actions for whatever
            situation might arise; this fully determines the agent behavior. A
            bidding strategy will determine the action of the agent will take at
            any stage of the bid generation and market-based scheduling
            processes, for every possible history and available market
            information to that stage.</Constant>
        </Annotation>
648 </EntityAnnotation>
        <EntityAnnotation>
650     <DataProperty URI="\&mxcsProtocol;strategy" />
        <Annotation annotationURI="\&rdfs;seeAlso">
652     <Constant>http://en.wikipedia.org/wiki/Strategy_(game_theory)</Constant>
        </Annotation>
654 </EntityAnnotation>
        <EntityAnnotation>
656     <DataProperty URI="\&mxcsProtocol;strategy" />
        <Annotation annotationURI="\&rdfs;seeAlso">
658     <Constant>http://mitpress.mit.edu/books/FLAOH/cbnhtml/glossary-S.html</
            Constant>
        </Annotation>
660 </EntityAnnotation>
        <Declaration>
662     <DataProperty URI="\&mxcsProtocol;strategy" />
        </Declaration>
664 <DataPropertyDomain>
        <DataProperty URI="\&mxcsProtocol;expiration" />
666     <Class URI="\&mxcsProtocol;PublicMessage" />
        </DataPropertyDomain>
668 <DataPropertyRange>
        <DataProperty URI="\&mxcsProtocol;expiration" />
670     <Datatype URI="\&xsd;long" />
        </DataPropertyRange>
672 <EntityAnnotation>
        <DataProperty URI="\&mxcsProtocol;expiration" />
674     <Annotation annotationURI="\&dc;relation">
        <Constant>http://sw.opencyc.org/concept/Mx4rvViQe5wpEbGdrcN5Y29ycA</
            Constant>
        </Annotation>
676 </EntityAnnotation>
678 <EntityAnnotation>
        <DataProperty URI="\&mxcsProtocol;expiration" />
680     <Annotation annotationURI="\&rdfs;comment">
        <Constant>The time in milliseconds a bid is valid in the target auction
            &#39;s order book. </Constant>
682     </Annotation>
        </EntityAnnotation>
684 <Declaration>
        <DataProperty URI="\&mxcsProtocol;expiration" />
686 </Declaration>
        <DataPropertyDomain>
688     <DataProperty URI="\&mxcsProtocol;valuation" />
        <Class URI="\&mxcsProtocol;PrivateMessage" />
690 </DataPropertyDomain>
        <DataPropertyDomain>
692     <DataProperty URI="\&mxcsProtocol;valuation" />
        <Class URI="\&mxcsProtocol;StateMessage" />
694 </DataPropertyDomain>

```

```

696     <DataPropertyRange>
        <DataProperty URI="\&mxcsProtocol;valuation" />
        <Datatype URI="\&xsd;double" />
698     </DataPropertyRange>
    <EntityAnnotation>
700     <DataProperty URI="\&mxcsProtocol;valuation" />
        <Annotation annotationURI="\&dc;relation">
702     <Constant>http://sw.opencyc.org/concept/Mx4rvWotk5wpEbGdrcN5Y29ycA/</
        Constant>
        </Annotation>
704     </EntityAnnotation>
    <EntityAnnotation>
706     <DataProperty URI="\&mxcsProtocol;valuation" />
        <Annotation annotationURI="\&rdfs;comment">
708     <Constant>The valuation is the monetary utility return of a good or a
        service. It is a subjective term that has value to one party may
        have no value to another. The valuation is the result of events in
        which someone estimates the amount that would be paid for a certain
        artifact either (a) if it were sold, called also reservation price (
        b) if a service should be bought, called also maximum willingness to
        pay.</Constant>
        </Annotation>
710     </EntityAnnotation>
    <EntityAnnotation>
712     <DataProperty URI="\&mxcsProtocol;valuation" />
        <Annotation annotationURI="\&rdfs;seeAlso">
714     <Constant>http://www.economypedia.com/wiki/index.php?title=Value/</
        Constant>
        </Annotation>
716     </EntityAnnotation>
    <Declaration>
718     <DataProperty URI="\&mxcsProtocol;valuation" />
    </Declaration>
720     <DataPropertyDomain>
        <DataProperty URI="\&mxcsProtocol;value" />
722     <Class URI="\&mxcsProtocol;MarketInformation" />
    </DataPropertyDomain>
724     <DataPropertyRange>
        <DataProperty URI="\&mxcsProtocol;value" />
726     <Datatype URI="\&xsd;double" />
    </DataPropertyRange>
728     <Declaration>
        <DataProperty URI="\&mxcsProtocol;value" />
730     </Declaration>
</Ontology>

```

C.3 State Message and Market Information

```

1 <?xml version='1.0' encoding='UTF-8'?>
2 <xsd:schema xmlns:xsd='http://www.w3.org/2001/XMLSchema'
3   xmlns:info='http://www.sormaproject.eu/message/ejsdl/beans'
4   xmlns:jsdl='http://schemas.ggf.org/jsdl/2005/11/jsdl'
5   xmlns:wsa='http://schemas.xmlsoap.org/ws/2004/03/addressing'
6   xmlns:sawsdl='http://www.w3.org/ns/sawsdl'
7     targetNamespace='http://www.sormaproject.eu/message/ejsdl/beans'
8     elementFormDefault='qualified'>
9
10  <xsd:import namespace='http://schemas.ggf.org/jsdl/2005/11/jsdl' schemaLocation='
11    jsdl.xsd' />
12  <xsd:import namespace='http://schemas.ggf.org/jsdl/2005/11/jsdl-posix'
13    schemaLocation='jsdl-posix.xsd' />
14
15  <!-- ===== -->
16  <!-- ===== StateMessage ===== -->
17  <!-- ===== -->
18
19  <xsd:complexType name='StateMessage_Type'>
20    <xsd:sequence>
21      <xsd:element ref='info:ScoringFunction' minOccurs='1' maxOccurs='1' />
22      <xsd:element ref='info:Parameters' minOccurs='1' maxOccurs='1' />
23    </xsd:sequence>
24    <xsd:attribute name='contractId' type='xsd:string' use='required'
25      sawsdl:modelReference='mxcsProtocol#contract' />
26  </xsd:complexType>
27
28  <xsd:complexType name='Parameters_Type'>
29    <xsd:sequence>
30      <xsd:element ref='info:valuation' minOccurs='1' maxOccurs='1' />
31      <xsd:element ref='info:timeToComplete' minOccurs='1' maxOccurs='1' />
32      <xsd:element ref='info:finalPrice' minOccurs='1' maxOccurs='1' />
33      <xsd:element ref='info:duration' minOccurs='1' maxOccurs='1' />
34    </xsd:sequence>
35  </xsd:complexType>
36
37  <xsd:element name='StateMessage' type='info:StateMessage_Type'
38    sawsdl:modelReference='mxcsProtocol#StateMessage' />
39  <xsd:element name='Parameters' type='info:Parameters_Type' sawsdl:modelReference='
40    mxcsProtocol#Parameters' />
41  <xsd:element name='ScoringFunction' type='xsd:string' sawsdl:modelReference='
42    mxcsProtocol#scoringFunction' />
43  <xsd:element name='valuation' type='xsd:double' sawsdl:modelReference='
44    mxcsProtocol#valuation' />
45  <xsd:element name='timeToComplete' type='xsd:double' sawsdl:modelReference='
46    mxcsProtocol#completionTime' />
47  <xsd:element name='finalPrice' type='xsd:string' sawsdl:modelReference='
48    mxcsProtocol#finalPrice' />
49  <xsd:element name='duration' type='xsd:double' sawsdl:modelReference='mxcsProtocol
50    #duration' />
51
52  <!-- ===== -->
53  <!-- ===== MarketInformation ===== -->
54  <!-- ===== -->
55
56  <xsd:complexType name='Queries_Type'>
57    <xsd:sequence>
58      <xsd:element ref='info:auctions' minOccurs='1' maxOccurs='1' />

```



```

49     </xsd:sequence>
    </xsd:complexType>

51 <xsd:complexType name='Response_Type'>
    <xsd:sequence>
53     <xsd:element ref='info:auctions' minOccurs='1' maxOccurs='1' />
    </xsd:sequence>
55 </xsd:complexType>

57 <xsd:complexType name='Auctions_Type'>
    <xsd:sequence>
59     <xsd:element ref='info:auction' minOccurs='0' />
    </xsd:sequence>
61 </xsd:complexType>

63 <xsd:complexType name='Auction_Type'>
    <xsd:sequence>
65     <xsd:element ref='info:marketinformation' minOccurs='0' />
    </xsd:sequence>
67     <xsd:attribute name='id' type='xsd:string' use='required' sawsdl:modelReference=
        'mxcsProtocol#id' />
    </xsd:complexType>

69 <xsd:complexType name='MarketInformation_Type'>
71     <xsd:sequence>
73     <xsd:element ref='info:lastNConsumerBids' minOccurs='0' maxOccurs='1' />
75     <xsd:element ref='info:lastNProviderBids' minOccurs='0' maxOccurs='1' />
77     <xsd:element ref='info:lastNClearingPrices' minOccurs='0' maxOccurs='1' />
79     <xsd:element ref='info:consumerbids' minOccurs='0' maxOccurs='1' />
81     <xsd:element ref='info:providerbids' minOccurs='0' maxOccurs='1' />
83     <xsd:element ref='info:clearingprices' minOccurs='0' maxOccurs='1' />
    </xsd:sequence>
    </xsd:complexType>

85 <xsd:complexType name='Value_Type'>
    <xsd:sequence>
87     <xsd:element ref='info:value' minOccurs='0' />
    </xsd:sequence>
    </xsd:complexType>

89 <xsd:element name='queries' type='info:Queries_Type' sawsdl:modelReference='
    mxcsProtocol#query' />
    <xsd:element name='response' type='info:Queries_Type' sawsdl:modelReference='
    mxcsProtocol#response' />
91 <xsd:element name='auctions' type='info:Auctions_Type' sawsdl:modelReference='
    mxcsProtocol#auction' />
    <xsd:element name='auction' type='info:Auction_Type' sawsdl:modelReference='
    mxcsProtocol#auction' />
93 <xsd:element name='marketinformation' type='info:MarketInformation_Type'
    sawsdl:modelReference='mxcsProtocol#marketinformation' />
    <xsd:element name='consumerbids' type='info:Value_Type' sawsdl:modelReference='
    mxcsProtocol#bid' />
95 <xsd:element name='providerbids' type='info:Value_Type' sawsdl:modelReference='
    mxcsProtocol#bid' />
    <xsd:element name='clearingprices' type='info:Value_Type' sawsdl:modelReference='
    mxcsProtocol#clearingPrice' />
    <xsd:element name='value' type='xsd:double' sawsdl:modelReference='mxcsProtocol#
    value' />
    <xsd:element name='lastNConsumerBids' type='xsd:integer' sawsdl:modelReference='
    mxcsProtocol#bid' />

```

```
97 | <xsd:element name='lastNProviderBids' type='xsd:integer' sawsdl:modelReference='  
    mxcsProtocol#bid' />  
    <xsd:element name='lastNClearingPrices' type='xsd:integer' sawsdl:modelReference='  
99 |     mxcsProtocol#clearingPrice' />  
    </xsd:schema>
```

Bibliography

- Ali, A., Anjum, A., Bunn, J., Cavanaugh, R., van Lingen, F., McClatchey, R., Mehmood, M. A., Newman, H., Steenberg, C., Thomas, M., and Willers, I. (2004). Predicting the resource requirements of a job submission. *Computing in High Energy Physics*, <http://cacr.library.caltech.edu/55/>.
- Altmann, J., Courcoubetis, C., Stamoulis, G., Dramitinos, M., Rayna, T., Risch, M., and Bannink, C. (2008). Gridcon: A market place for computing resources. In Altmann, J., Neumann, D., and Fahringer, T., editors, *Grid Economics and Business Models*, volume 5206 of *Lecture Notes in Computer Science*, pages 185–196. Springer Berlin / Heidelberg. 10.1007/978-3-540-85485-2_15.
- Amar, L., Mu’alem, A., and Stosser, J. (2008). On the importance of migration for fairness in online grid markets. In *Proceedings of the 2008 9th IEEE/ACM International Conference on Grid Computing-Volume 00*, pages 65–74. IEEE Computer Society.
- Amazon (2009). Amazon elastic compute cloud. an introduction to spot instances, api version 2009-11-30.
- Amazon (2010a). Amazon instance types, <http://aws.amazon.com/ec2/instance-types/>.
- Amazon (2010b). Amazon web services – case studies. <http://aws.amazon.com/solutions/case-studies/>.
- An, B., Lesser, V., Irwin, D., and Zink, M. (2010). Automated negotiation with decommitment for dynamic resource allocation in cloud computing. *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*.
- Andretto, P., Androozzi, S., Ghiselli, A., Marzolla, M., Venturi, V., and Zangrando, L. (2010). Standards-based job management in grid systems. *Journal of Grid Computing*, 8:19–45.
- Andrieux, A. et al. (2007). Web services agreement specification (WS-Agreement).
- Anjomshoaa, A., Brisard, F., Drescher, M., and Fellows, D. (2005). Job Submission Description Language (JSDL) Specification, Version 1.0. In *Global Grid Forum (GGF)*, page 72.
- Anthony, P. and Jennings, N. (2003). Developing a bidding agent for multiple heterogeneous auctions. *ACM Transactions on Internet Technology (TOIT)*, 3(3):185–217.
- Armando, A., Carbone, R., Compagna, L., Cuellar, J., and Tobarra, L. (2008). Formal analysis of saml 2.0 web browser single sign-on: breaking the saml-based single sign-

- on for google apps. In *Proceedings of the 6th ACM workshop on Formal methods in security engineering, FMSE '08*, pages 1–10, New York, NY, USA. ACM.
- Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al. (2009). Above the clouds: A Berkeley View of Cloud Computing. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28*.
- Auyoung, A., Buonadonna, P., Chun, B. N., Ng, C., Parkes, D. C., Shneidman, J., Snoeren, A. C., and Vahdat, A. (2009). Two auction-based resource allocation environments: Design and experience. In Buyya, R. and Bubendorfer, K., editors, *Market Oriented Grid and Utility Computing*, chapter 23. Wiley.
- AuYoung, A., Chun, B., Ng, C., Parkes, D., Vahdat, A., and Snoeren, A. (2007). Practical market-based resource allocation. *University of California, San Diego, Tech. Rep.*
- AuYoung, A., Grit, L., Wiener, J., and Wilkes, J. (2006). Service contracts and aggregate utility functions. *Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing*, pages 119–131.
- Axelrod, R. and Hamilton, W. (1981). The evolution of cooperation. *Science*, 211(4489):1390.
- Baker, M. and Buyya, R. (1999). Cluster computing at a glance. *High Performance Cluster Computing*, 1:3–47.
- Bapna, R., Goes, P., Gupta, A., and Jin, Y. (2004). User heterogeneity and its impact on electronic auction market design: An empirical exploration. *MIS Quarterly*, 28(1):21–43.
- Barak, A., Shiloh, A., and Amar, L. (2005). An organizational grid of federated mosix clusters. *Cluster Computing and the Grid, IEEE International Symposium on*, 1:350–357.
- Barto, A. and Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379.
- Bartolini, C., Priest, C., and Jennings, N. (2005). A software framework for automated negotiation. *Software Engineering for Multi-Agent Systems III: Research Issues and Practical Applications*, pages 213–235.
- Bauer, B. and Odell, J. (2005). Uml 2.0 and agents: how to build agent-based systems with the new uml standard. *Engineering Applications of Artificial Intelligence*, 18(2):141–157.
- Becker, M., Borisssov, N., Deora, V., Rana, O., and Neumann, D. (2008). Using k-Pricing for Penalty Calculation in Grid Market. *Hawaii International Conference on System Sciences, Proceedings of the 41st Annual*, pages 97–97.

- Bergemann, D. and Pesendorfer, M. (2007). Information structures in optimal auctions. *Journal of Economic Theory*, 137(1):580 – 609.
- Bergenti, F. and Poggi, A. (2002). LEAP: A FIPA Platform for Handheld and Mobile Devices. *LECTURE NOTES IN COMPUTER SCIENCE*, pages 436–446.
- Berl, A., Gelenbe, E., Di Girolamo, M., Giuliani, G., De Meer, H., Dang, M. Q., and Pentikousis, K. (2010). Energy-efficient cloud computing. *The Computer Journal*, 53(7):1045–1051.
- Bernstein, P. A. (1996). Middleware: a model for distributed system services. *Commun. ACM*, 39(2):86–98.
- Blau, B., Kraemer, J., Conte, T., and van Dinther, C. (2009). Service value networks. *CEC '09: Proceedings of the 2009 IEEE Conference on Commerce and Enterprise Computing*, pages 194–201.
- Blume, L. (2010). Robustness and fragility of markets: Research at the interface of economics and computer science. Technical report, “Grand Challenge” White Papers for Future Research in the Social, Behavioral & Economic Sciences, National Science Foundation.
- Bordini, R. H., Braubach, L., Dastani, M., Fallah-Seghrouchni, A. E., Gómez-Sanz, J. J., Leite, J., O’Hare, G. M. P., Pokahr, A., and Ricci, A. (2006). A survey of programming languages and platforms for multi-agent systems. *Informatika (Slovenia)*, 30(1):33–44.
- Borissov, N. (2009). Q-Strategy: Automated Bidding and Convergence in Computational Markets. In *Twenty-first Innovative Applications of Artificial Intelligence (IAAI) Conference collocated with IJCAI, Pasadena, California*, pages 54–59. Twenty-first IAAI Conference on Artificial Intelligence, Pasadena, California.
- Borissov, N., Blau, B., and Neumann, D. (2008a). Semi-automated provisioning and usage of configurable services. *16th European Conference on Information Systems (ECIS), 2008, Galway, Ireland*, page 19411952.
- Borissov, N., Brunner, R., Neumann, D., Freitag, F., Navarro, L., and Weinhardt, C. (2009a). Fostering efficiency of computational resource allocation - integrating information services into markets. In *17th European Conference on Information Systems (ECIS-2009)*, Verona.
- Borissov, N., Caton, S., Rana, O., and Levine, A. (2009b). Message Protocols for Provisioning and Usage of Computing Services. *Proceedings of the 6th International Workshop on Grid Economics and Business Models*, 5745/2009:170.
- Borissov, N., Neumann, D., and Weinhardt, C. (2010). Automated bidding in computational markets: an application in market-based allocation of computing services. *Autonomous Agents and Multi-Agent Systems*, 21(2):115–142.

- Borissov, N., Nimis, J., Wirström, N., and Rasmusson, L. (2008b). D4.2 - bid and offer generator prototype. Technical report, Technical Report. IST-FP6-034286 SORMA.
- Borissov, N. and Wirström, N. (2008). Q-strategy: A bidding strategy for market-based allocation of grid services. In Meersman, R. and Tari, Z., editors, *On the Move to Meaningful Internet Systems: OTM 2008*, volume 5331 of *Lecture Notes in Computer Science*, pages 744–761. Springer Berlin / Heidelberg.
- Borodin, A. and El-Yaniv, R. (1998). *Online computation and competitive analysis*. Cambridge University Press, New York, NY, USA.
- Bowling, M. H. (2000). Convergence problems of general-sum multiagent reinforcement learning. In *ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning*, pages 89–94, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- Bratman, J., Fabbri, D., and Zimmerman, A. (2009). Reinforcement learning approach to managing distributed data processing tasks in wireless sensing networks. Technical report, In EECS 545 Machine Learning. University of Michigan.
- Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., and Mylopoulos, J. (2004). Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8:203–236. 10.1023/B:AGNT.0000018806.20944.ef.
- Broberg, J., Venugopal, S., and Buyya, R. (2008). Market-oriented grids and utility computing: The state-of-the-art and future directions. *Journal of Grid Computing*, 6(3):255–276.
- Brooks, H., Gazzale, R. S., MacKie Mason, J. K., and Durfee, E. H. (2003). Improving learning performance by applying economic knowledge. In *EC '03: Proceedings of the 4th ACM conference on Electronic commerce*, pages 252–253, New York, NY, USA. ACM.
- Brown, D. and Reams, C. (2010). Toward energy-efficient computing. *Communications of the ACM*, 53(3):50–58.
- Brunner, R., Freitag, F., and Navarro, L. (2008). Towards the development of a decentralized market information system: Requirements and architecture. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–7.
- Brynjolfsson, E., Hofmann, P., and Jordan, J. (2010). Cloud computing and electricity: beyond the utility model. *Communications of the ACM*, 53(5):32–34.
- Buyya, R. (2002). *Economic-based Distributed Resource Management and Scheduling for Grid Computing*. PhD thesis, Monash University.
- Buyya, R., Abramson, D., Giddy, J., and Stockinger, H. (2002). Economic models for

- resource management and scheduling in grid computing. *Concurrency and computation: practice and experience*, 14(13-15):1507–1542.
- Buyya, R., Abramson, D., and Venugopal, S. (2005). The grid economy. *Proceedings of the IEEE*, 93(3):698–714.
- Buyya, R., Yeo, C., Venugopal, S., Broberg, J., and Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599–616.
- Cacciari, C., Mallmann, D., Zsigri, C., DAndria, F., Hagemeyer, B., Rumpl, A., Ziegler, W., and Martrat, J. (2010). Sla-based management of software licenses as web service resources in distributed environments. In Altmann, J. and Rana, O., editors, *Economics of Grids, Clouds, Systems, and Services*, volume 6296 of *Lecture Notes in Computer Science*, pages 78–92. Springer Berlin / Heidelberg. 10.1007/978-3-642-15681-6₆.
- Cai, K., Gerding, E., McBurney, P., Niu, J., Parsons, S., and Phelps, S. (2009). Overview of cat: A market design competition. version 2.0. Technical report, Technical Report ULCS-09-005, Department of Computer Science, University of Liverpool, Liverpool, UK.
- Campbell, J., Lo, A., MacKinlay, A., and Whitelaw, R. (1997). *The econometrics of financial markets*. Princeton University Press Princeton, NJ.
- Campbell, R., Gupta, I., Heath, M., Ko, S., Kozuch, M., Kunze, M., Kwan, T., Lai, K., Lee, H., Lyons, M., et al. (2009). Open Cirrus™ Cloud Computing Testbed: Federated Data Centers for Open Source Systems and Services Research. *Proceedings of the USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*.
- Catteddu, D. and Hogben, G. (2009). Cloud Computing: benefits, risks and recommendations for information security. Technical report, Technical Report. European Network and Information Security Agency.
- Chacin, P., Leon, X., Brunner, R., Freitag, F., and Navarro, L. (2008). Core Services For Grid Markets. *From Grids to Service and Pervasive Computing*, pages 205–215.
- Chander, P. and Tulkens, H. (2006). Exchange processes, the core and competitive allocations. In Chander, P., Drze, J., Lovell, C. K., and Mintz, J., editors, *Public goods, environmental externalities and fiscal competition*, pages 64–79. Springer US. 10.1007/978-0-387-25534-7₄.
- Chard, K., Bubendorfer, K., Caton, S., and Rana, O. (2011). Social cloud computing: A vision for socially motivated resource sharing. *IEEE Transactions on Services Computing*, forthcoming.
- Chavez, A. and Maes, P. (1996). Kasbah: An agent marketplace for buying and selling

- goods. *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, 31:40.
- Cheliotis, G., Kenyon, C., Buyya, R., and Melbourne, A. (2005). 10 Lessons from Finance for Commercial Sharing of IT Resources. *Peer-to-Peer Computing: The Evolution of a Disruptive Technology*, Idea Group Publishing, pages 244–264.
- Cheng, J. and Wellman, M. (1998). The WALRAS algorithm: A convergent distributed implementation of general equilibrium outcomes. *Computational Economics*, 12(1):1–24.
- Chevaleyre, Y., Dunne, P., Endriss, U., Lang, J., Lemaitre, M., Maudet, N., Padget, J., Phelps, S., Rodriguez-Aguilar, J., and Sousa, P. (2006). Issues in multiagent resource allocation. *Special Issue: Hot Topics in European Agent Research II Guest Editors: Andrea Omicini*, 30:3–31.
- Chevaleyre, Y., Endriss, U., Estivie, S., and Maudet, N. (2008). Multiagent resource allocation in k-additive domains: Preference representation and complexity. *Annals of Operations Research*, 163(1):49–62.
- Clemons, E. (2009). Business models for monetizing internet applications and web sites: Experience, theory, and predictions. *J. Manage. Inf. Syst.*, 26(2):15–41.
- Cliff, D. (2006). Zip60: an enhanced variant of the zip trading algorithm. In *CEC-EEE '06: Proceedings of the The 8th IEEE International Conference on E-Commerce Technology and The 3rd IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services*, page 15, Washington, DC, USA. IEEE Computer Society.
- Cliff, D. and Bruten, J. (1997). Zero is not enough: On the lower limit of agent intelligence for continuous double auction markets. Technical report, HP Labs Technical Reports, HPL-97-141.
- Collins, J., Youngdahl, B., Jamison, S., Mobasher, B., and Gini, M. (1998). A market architecture for multi-agent contracting. In *Proceedings of the second international conference on Autonomous agents*, AGENTS '98, pages 285–292, New York, NY, USA. ACM.
- Cotton, I. (1975). Microeconomics and the market for computer services. *ACM Computing Surveys (CSUR)*, 7(2):111.
- Craig, R., Frazier, J., Jacknis, N., Murphy, S., Purcell, C., Spencer, P., and Stanley, J. (2009). Cloud computing in the public sector: Public managers guide to evaluating and adopting cloud computing. *White Paper, Cisco Internet Business Solutions Group (IBSG)*.
- Cusumano, M. (2008). The changing software business: Moving from products to services. *Computer*, pages 20–27.

- Das, R., Hanson, J. E., Kephart, J. O., and Tesauro, G. (2001). Agent-human interactions in the continuous double auction. In *IJCAI'01: Proceedings of the 17th international joint conference on Artificial intelligence*, pages 1169–1176, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- Dash, R., Jennings, N., and Parkes, D. (2003). Computational-mechanism design: a call to arms. *Intelligent Systems, IEEE [see also IEEE Intelligent Systems and Their Applications]*, 18(6):40–47.
- De Boer, L., Labro, E., and Morlacchi, P. (2001). A review of methods supporting supplier selection. *European Journal of Purchasing & Supply Management*, 7(2):75–89.
- Dearden, R., Friedman, N., and Russell, S. (1998). Bayesian q-learning. In *Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, AAAI '98/IAAI '98, pages 761–768, Menlo Park, CA, USA. American Association for Artificial Intelligence.
- Demchenko, Y., Gommans, L., and Laat, C. d. (2007). Using saml and xacml for complex resource provisioning in grid based applications. In *Proceedings of the Eighth IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 183–187, Washington, DC, USA. IEEE Computer Society.
- Diamantini, C., Potena, D., and Cellini, J. (2007). UDDI registry for Knowledge Discovery in Databases services. in *Proc. of the International Symposium on Collaborative Technologies and Systems, IEEE*, pages 321–328.
- DMTF (2010a). Cloud incubator. Distributed Management Task Force.
- DMTF (2010b). *Common Information Model (CIM) v2.26*. Distributed Management Task Force (DMTF), <http://www.dmtf.org/standards/cim>.
- Dong, F. and Akl, S. (2006). Scheduling algorithms for grid computing: State of the art and open problems. Technical report, School of Computing, Queens University, Kingston, Ontario.
- Doorenbos, R. B., Etzioni, O., and Weld, D. S. (1997). A scalable comparison-shopping agent for the world-wide web. In *AGENTS '97: Proceedings of the first international conference on Autonomous agents*, pages 39–48, New York, NY, USA. ACM.
- Dorn, C., Schall, D., and Dustdar, S. (2009). Context-aware adaptive service mashups. In *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific*, pages 301–306.
- Dunsavage, K., Ott, D., Raghu, R., and Smith, S. (2010). Software Applications in the Clouds: Design, Deployment, and Intel Platforms. *White Paper, Intel*.
- EGEE (2009). Enabling grids for e-science.

- Ellert, M., Grønager, M., Konstantinov, A., Kónya, B., Lindemann, J., Livenson, I., Nielsen, J., Niinimäki, M., Smirnova, O., and Wäänänen, A. (2007). Advanced resource connector middleware for lightweight computational grids. *Future Generation Computer Systems*, 23(2):219 – 240.
- Elmroth, E. and Gardfjall, P. (2005). Design and evaluation of a decentralized system for grid-wide fairshare scheduling. In *e-Science and Grid Computing, 2005. First International Conference on*, pages 9 pp. –229.
- Elmroth, E., Hernández, F., Tordsson, J., and Stberg, P.-O. (2008). Designing service-based resource management tools for a healthy grid ecosystem. In Wyrzykowski, R., Dongarra, J., Karczewski, K., and Wasniewski, J., editors, *Parallel Processing and Applied Mathematics*, volume 4967 of *Lecture Notes in Computer Science*, pages 259–270. Springer Berlin / Heidelberg. 10.1007/978-3-540-68111-3₂₈.
- Endriss, U. and Maudet, N. (2004). On the communication complexity of multilateral trading. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 622–629, Washington, DC, USA. IEEE Computer Society.
- Engelbrecht-Wiggans, R. (1980). Auctions and bidding models: A survey. *Management Science*, 26(2):119–142.
- Erev, I. and Roth, A. (1998). Predicting how people play games: Reinforcement learning in experimental games with unique, mixed strategy equilibria. *The American Economic Review*, 88(4):848–881.
- Even-Dar, E., Mannor, S., and Mansour, Y. (2003). Action Elimination and Stopping Conditions for Reinforcement Learning. *Proc. of the 20th International Conference on Machine Learning*, 20(1):162.
- Even-Dar, E. and Mansour, Y. (2004). Learning Rates for Q-learning. *The Journal of Machine Learning Research*, 5:1–25.
- Fasli, M. and Michalakopoulos, M. (2008). e-Game: A platform for developing auction-based market simulations. *Decision Support Systems*, 44(2):469–481.
- Feigenbaum, J., Parkes, D. C., and Pennock, D. M. (2009). Computational challenges in e-commerce. *Commun. ACM*, 52(1):70–74.
- Feitelson, D. (2010). Parallel workloads archive. <http://www.cs.huji.ac.il/labs/parallel/workload>.
- Feitelson, D. G., Rudolph, L., and Schwiegelshohn, U. (2005). Parallel job scheduling a status report. In Feitelson, D., Rudolph, L., and Schwiegelshohn, U., editors, *Job Scheduling Strategies for Parallel Processing*, volume 3277 of *Lecture Notes in Computer Science*, pages 1–16. Springer Berlin / Heidelberg. 10.1007/11407522₁.

- Feldman, M., Lai, K., and Zhang, L. (2009). The Proportional-Share Allocation Market for Computational Resources. *IEEE Transactions on Parallel and Distributed Systems*, 20(8):1075–1088.
- Feller, M., Foster, I., and Martin, S. (2007). GT4 GRAM: A functionality and performance study. *TeraGrid Conference*.
- FIPA (2002a). FIPA ACL Message Structure Specification. Technical report.
- FIPA (2007). Agentuml. <http://www.auml.org/>.
- FIPA, T. (2002b). Fipa abstract architecture specification. Technical report, Foundation for Intelligent Physical Agents.
- Foley, D. K. (2010). What’s wrong with the fundamental existence and welfare theorems? *Journal of Economic Behavior & Organization*, In Press, Accepted Manuscript:–.
- Fornara, N., Vigano, F., and Colombetti, M. (2007). Agent communication and artificial institutions. *Autonomous Agents and Multi-Agent Systems*, 14:121–142. 10.1007/s10458-006-0017-8.
- Foster, I. (2002). What is the grid? a three point checklist. *GRID today*, 1(6):22–25.
- Foster, I., Czajkowski, K., Ferguson, D., Frey, J., Graham, S., Maguire, T., Snelling, D., and Tuecke, S. (2005). Modeling and managing state in distributed systems: The role of OGSi and WSRF. *Proceedings of the IEEE*, 93(3):604–612.
- Foster, I., Jennings, N., and Kesselman, C. (2004). Brain meets brawn: Why grid and agents need each other. *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, 1:15.
- Foster, I. and Kesselman, C. (1997). Globus: A metacomputing infrastructure toolkit. *International Journal of High Performance Computing Applications*, 11(2):115.
- Foster, I., Kesselman, C., Nick, J., and Tuecke, S. (2002). Grid services for distributed system integration. *Computer*, pages 37–46.
- Foster, I., Kishimoto, H., Savva, A., Berry, D., Djaoui, A., Grimshaw, A., Horn, B., Maciel, F., Siebenlist, F., Subramaniam, R., et al. (2006). The Open Grid Services Architecture, Version 1.5. *Open Grid Forum, GFD*, 80.
- Foster, I., Parastatidis, S., Watson, P., and McKeown, M. (2009). How do I model state? Let me count the ways. *Queue*, 7(2):54–55.
- Foster, I., Zhao, Y., Raicu, I., and Lu, S. (2008). Cloud Computing and Grid Computing 360-Degree Compared. *Grid Computing Environments Workshop*, pages 1–10.
- Friedman, D. (1993). The double auction market institution: A survey. *The Double Auction Market: Institutions, Theories, and Evidence*, pages 3–25.

- Gagliano, R., Fraser, M., and Schaefer, M. (1995). Auction allocation of computing resources. *Communications of the ACM*, 38(6):88–102.
- Galán, F., Sampaio, A., Rodero-Merino, L., Loy, I., Gil, V., and Vaquero, L. (2009). Service specification in cloud environments based on extensions to open standards. In *Proceedings of the Fourth International ICST Conference on COMMunication System softWARE and middlewaRE*, pages 1–12. ACM.
- Gasević, D., Kaviani, N., and Milanović, M. (2009). Ontologies and software engineering. In Staab, S. and Rudi Studer, D., editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 593–615. Springer Berlin Heidelberg. 10.1007/978-3-540-92673-3_27.
- Geelan, J. (2009). Twenty experts define cloud computing.
- Gjerstad, S. (2003). The strategic impact of pace in double auction bargaining. *Microeconomics*, EconWPA.
- Gjerstad, S. (2007). The competitive market paradox. *Journal of Economic Dynamics and Control*, 31(5):1753–1780.
- Gjerstad, S. and Dickhaut, J. (1998). Price formation in double auctions. *Games and Economic Behavior*, 22(1):1–29.
- Gode, D. and Sunder, S. (1993). Allocative efficiency of markets with zero-intelligence traders: Market as a partial substitute for individual rationality. *The Journal of Political Economy*, 101(1):119–137.
- Gold, N., Mohan, A., Knight, C., and Munro, M. (2004). Understanding service-oriented software. *IEEE software*, pages 71–77.
- Gomes, E. R. and Kowalczyk, R. (2007). Reinforcement learning with utility-aware agents for market-based resource allocation. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–3, New York, NY, USA. ACM.
- Goodale, T., Jha, S., Kaiser, H., Kielmann, T., al Kleijer, P., Merzky, A., Shalf, J., and Smith, C. (2008). A Simple API for Grid Applications (SAGA). *OGF Document Series 90*.
- Greenberg, A., Hamilton, J., Maltz, D., and Patel, P. (2008). The cost of a cloud: research problems in data center networks. *ACM SIGCOMM Computer Communication Review*, 39(1):68–73.
- Greenwald, A., Jennings, N. R., and Stone, P. (2003). Guest editors' introduction: Agents and markets. *IEEE Intelligent Systems*, 18:12–14.
- Greenwald, A. R., Kephart, J. O., and Tesauro, G. J. (1999). Strategic pricebot dynamics. In *EC '99: Proceedings of the 1st ACM conference on Electronic commerce*, pages 58–67, New York, NY, USA. ACM.

- Grosu, D. and Das, A. (2006). Auctioning resources in grids: model and protocols: Research articles. *Concurrency and Computation: Practice and Experience*, 18(15):1909–1927.
- He, M., Leung, H., and Jennings, N. (2003). A fuzzy-logic based bidding strategy for autonomous agents in continuous double auctions. *IEEE Transactions on Knowledge and Data Engineering*, 15(6):1345–1363.
- Helsing, A., Thome, M., Wright, T., Technol, B., and Cambridge, M. (2004). Cougaar: a scalable, distributed multi-agent architecture. *IEEE INTERNATIONAL CONFERENCE ON SYSTEMS MAN AND CYBERNETICS*, 2:1910–1917.
- Hevner, A. R., March, S. T., Park, J., and Ram, S. (2004). Design science in information systems research. *MIS Quarterly*, 28(1):75–105.
- Heydenreich, B., Müller, R., and Uetz, M. (2010). Mechanism design for decentralized online machine scheduling. *Operations Research*, 58(2):445–457.
- Hommes, C. (2006). Heterogeneous agent models in economics and finance. *Handbook of computational economics*, pages 1109–1186.
- Howden, N., Ronnquist, R., Hodgson, A., and Lucas, A. (2001). JACK Intelligent Agents-Summary of an Agent Infrastructure. *Proceedings of the 5th International Conference on Autonomous Agents*.
- Hu, J. and Wellman, M. P. (1998). Online learning about other agents in a dynamic multiagent system. In *AGENTS '98: Proceedings of the second international conference on Autonomous agents*, pages 239–246, New York, NY, USA. ACM.
- ISACA (2009). Cloud computing: Business benefits with security, governance and assurance perspectives. *Information Systems Audit and Control Association*.
- Iyer, K. and Huhns, M. N. (2009). Negotiation criteria for multiagent resource allocation. *Knowl. Eng. Rev.*, 24(2):111–135.
- Jain, R. (1991). *The art of computer systems performance analysis*. Wiley-India.
- Jansen, W. and Grance, T. (2011). Guidelines on security and privacy in public cloud computing. *Draft Special Publication 800-144, National Institute of Standards and Technology*, page 60.
- Jennings, N., Sycara, K., and Wooldridge, M. (1998). A roadmap of agent research and development. *Autonomous agents and multi-agent systems*, 1(1):7–38.
- Jennings, N. R. (2001). An agent-based approach for building complex software systems. *Commun. ACM*, 44(4):35–41.
- Jensen, M., Schwenk, J., Gruschka, N., and Iacono, L. L. (2009). On technical security issues in cloud computing. *Cloud Computing, IEEE International Conference on*, 0:109–116.

- Jiang, A. and Leyton-Brown, K. (2007). Bidding agents for online auctions with hidden bids. *Machine Learning*, 67:117–143.
- Jordan, P. R. and Wellman, M. P. (2010). Designing an ad auctions game for the trading agent competition. In Aalst, W., Mylopoulos, J., Sadeh, N. M., Shaw, M. J., Szyperski, C., David, E., Gerding, E., Sarne, D., and Shehory, O., editors, *Agent-Mediated Electronic Commerce. Designing Trading Strategies and Mechanisms for Electronic Markets*, volume 59 of *Lecture Notes in Business Information Processing*, pages 147–162. Springer Berlin Heidelberg. 10.1007/978-3-642-15117-0_11.
- Jordan, P. R., Wellman, M. P., and Balakrishnan, G. (2010). Strategy and mechanism lessons from the first ad auctions trading agent competition. In *Proceedings of the 11th ACM conference on Electronic commerce, EC '10*, pages 287–296, New York, NY, USA. ACM.
- Josang, A., Ismail, R., and Boyd, C. (2007). A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43(2):618–644.
- Kaelbling, L., Littman, M., and Moore, A. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4(237-285):102–138.
- Kalagnanam, J. and Parkes, D. (2004). Auctions, bidding and exchange design. *Handbook of Quantitative Supply Chain Analysis: Modeling in the E-Business Era*, ed. D. Simchi-Levi, SD Wu, Z. Shen, pages 143–212.
- Kant, U. and Grosu, D. (2005). Double auction protocols for resource allocation in grids. *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC05)*, pages 366–371.
- Kephart, J., Hanson, J., and Greenwald, A. (2000). Dynamic pricing by software agents. *Computer Networks*, 32(6):731–752.
- Kephart, J. and Walsh, W. (2004a). An artificial intelligence perspective on autonomic computing policies. *Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 3–12.
- Kephart, J. O. and Chess, D. M. (2003). The vision of autonomic computing. *Computer*, 36(1):41–50.
- Kephart, J. O. and Walsh, W. E. (2004b). An artificial intelligence perspective on autonomic computing policies. *Policies for Distributed Systems and Networks, IEEE International Workshop on*, 0:3.
- Kiekintveld, C., Miller, J., Jordan, P., Callender, L., and Wellman, M. (2009). Forecasting market prices in a supply chain game. *Electronic Commerce Research and Applications*, 8(2):63–77.
- Kiselev, I. and Alhajj, R. (2009). A multiagent approach to adaptive continuous analysis

- of streaming data in complex uncertain environments. *Data Mining and Multi-agent Integration, Springer US*, pages 201–218.
- Kopecky, J., Vitvar, T., Bournez, C., and Farrell, J. (2007). Sawsdl: Semantic annotations for wsdl and xml schema. *IEEE Internet Computing*, 11(6):60–67.
- Koutsoupias, E. and Papadimitriou, C. (2009). Worst-case equilibria. *Computer Science Review*, 3(2):65 – 69.
- Krishna, V. and Perry, M. (1998). Efficient mechanism design. Game theory and information, EconWPA.
- Ku, G., Malhotra, D., and Murnighan, J. K. (2005). Towards a competitive arousal model of decision-making: A study of auction fever in live and internet auctions. *Organizational Behavior and Human Decision Processes*, 96(2):89 – 103.
- Kübert, R., Katsaros, G., and Wang, T. (2011). A restful implementation of the ws-agreement specification. In *Proceedings of the Second International Workshop on RESTful Design, WS-REST '11*, pages 67–72, New York, NY, USA. ACM.
- Labrou, Y., Finin, T., and Peng, Y. (1999). Agent communication languages: the current landscape. *Intelligent Systems and their Applications, IEEE*, 14(2):45 –52.
- Lahaie, S., Pennock, D., Saberi, A., and Vohra, R. (2007). Sponsored search auctions. *Algorithmic Game Theory*, pages 699–716.
- Lai, K. (2005). Markets are dead, long live markets. *ACM SIGecom Exchanges*, 5(4):10.
- Lai, K., Rasmusson, L., Adar, E., Zhang, L., and Huberman, B. (2005). Tycoon: An implementation of a distributed, market-based resource allocation system. *Multiagent and Grid Systems*, 1(3):169–182.
- Lambrecht, A. and Skiera, B. (2006). Paying too much and being happy about it: Existence, causes, and consequences of tariff-choice biases. *Journal of marketing research*, 43(2):212–223.
- Lathem, J., Gomadam, K., and Sheth, A. (2007). Sa-rest and (s)mashups : Adding semantics to restful services. pages 469 –476.
- Laure, E., Fisher, S., Frohner, A., Grandi, C., Kunszt, P., Krenek, A., Mulmo, O., Pacini, F., Prelz, F., White, J., et al. (2006). Programming the Grid with gLite. *Computational Methods in Science and Technology*, 12(1):33–45.
- Lenk, A., Klems, M., Nimis, J., Tai, S., Karlsruhe, F., and Sandholm, T. (2009). Whats Inside the Cloud? An Architectural Map of the Cloud Landscape. *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, pages 23–31.
- Levine, D. K. (2010). Virtual model validation for economics. Technical report, “Grand Challenge” White Papers for Future Research in the Social, Behavioral & Economic Sciences, National Science Foundation.

- Li, J. and Yahyapour, R. (2006). Learning-based negotiation strategies for grid scheduling. in *Proceedings of the International Symposium on Cluster Computing and the Grid (CCGRID2006)*, pages 576–583.
- Lin, R. and Kraus, S. (2010). Can automated agents proficiently negotiate with humans? *Communications of the ACM*, 53(1):78–88.
- Lind, J. (2001). Issues in agent-oriented software engineering. In *First international workshop, AOSE 2000 on Agent-oriented software engineering*, pages 45–58, Secaucus, NJ, USA. Springer-Verlag New York, Inc.
- Lingrand, D., Montagnat, J., Martyniak, J., and Colling, D. (2009). Analyzing the EGEE production grid workload: application to jobs submission optimization. *Job Scheduling Strategies for Parallel Processing: 14th International Workshop*, page 37.
- Liu, C. and Foster, I. (2004). A Constraint Language Approach to Matchmaking. *14th Int. Workshop on Research Issues on Data Engineering: Web Services for E-Commerce and E-Government Applications*, pages 7–14.
- Lomuscio, A. R., Wooldridge, M., and Jennings, N. (2003). A classification scheme for negotiation in electronic commerce. *Group Decision and Negotiation*, 12:31–56. 10.1023/A:1022232410606.
- Love, N., Hinrichs, T., and Genesereth, M. (2006). General game playing: Game description language specification. Technical report, Technical Report LG-2006-01, Stanford Logic Group.
- Lubin, B., Juda, A. I., Cavallo, R., Lahaie, S., Shneidman, J., and Parkes, D. C. (2008). Ice: an expressive iterative combinatorial exchange. *J. Artif. Int. Res.*, 33(1):33–77.
- Lubin, B., Parkes, D. C., Kephart, J., and Das, R. (2009). Expressive Power-Based Resource Allocation for Data Centers. In *Twenty-First International Joint Conference on Artificial Intelligence (IJCAI'09)*, pages 1451–1456.
- Macías, M., Rana, O., Smith, G., Guitart, J., and Torres, J. (2008). Maximizing revenue in grid markets using an economically enhanced resource manager. *Concurrency and Computation: Practice and Experience*, pages 1532–0626.
- MacKie-Mason, J. and Wellman, M. (2006). Automated Markets and Trading Agents. *Handbook of Computational Economics*, 2:1381–1431.
- Mascardi, V., Cordì, V., and Rosso, P. (2007). A comparison of upper ontologies. *Agenti e industria: Applicazioni tecnologiche degli agenti software, WOA07*, pages 24–25.
- Maximilien, E. M., Wilkinson, H., Desai, N., and Tai, S. (2007). A domain-specific language for web apis and services mashups. In *ICSOC '07: Proceedings of the 5th international conference on Service-Oriented Computing*, pages 13–26, Berlin, Heidelberg. Springer-Verlag.

- McAfee, R. and McMillan, J. (1987). Auctions and bidding. *Journal of Economic Literature*, 25(2):699–738.
- McGough, A. and Savva, A. (2008). Implementation and Interoperability Experiences with the Job Submission Description Language (JSDL) 1.0 (Draft 009). OGF JSDL Working Group, <http://forge.ggf.org/sf/go/doc15267?nav=1>.
- Meinl, T. and Blau, B. (2009). Web service derivatives. In *Proceedings of the 18th international conference on World wide web, WWW '09*, pages 271–280, New York, NY, USA. ACM.
- Mell, P. and Grance, T. (2009a). Effectively and securely using the cloud computing paradigm. *NIST, Information Technology Laboratory*, pages 1–92.
- Mell, P. and Grance, T. (2009b). The NIST Definition of Cloud Computing. National Institute of Standards and Technology. *Information Technology Laboratory, Version 15*.
- Mendelson, H. (1985). Pricing computer services: queueing effects. *Communications of the ACM*, 28(3):321.
- Michalk, W. and Blau, B. (2010). Risk in Agreement Networks Decision Support for Service-Intermediaries. *Journal of Information Systems and e-Business Management (ISeBM)*, forthcoming.
- Michalk, W., Filipova-Neumann, L., Blau, B., and Weinhardt, C. (2011). Reducing Risk or Increasing Profit? Provider Decisions in Agreement Networks. *Service Science*, forthcoming.
- Microsystems, S. (2009). Introduction to cloud computing architecture. *White Paper. 1st Edition, June*.
- Milgrom, P. and Weber, R. (1982). A theory of auctions and competitive bidding. *Econometrica*, 50(5):1089–1122.
- Milojicic, D. et al. (1998). MASIF: The OMG mobile agent system interoperability facility. *Personal and Ubiquitous Computing*, 2:117–129.
- Mowbray, M. (2009). The Fog over the Grimpen Mire: Cloud Computing and the Law. *Script-ed Journal of Law, Technology and Society*, 6, no.1(1).
- Mukherji, A. (2008). Stability of a competitive economy: A reconsideration. *International Journal of Economic Theory*, 4(2):317–336.
- Müller, K.-R., Mika, S., Rätsch, G., Tsuda, S., and Schölkopf, B. (2001). An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12(2):181–202.
- Myerson, R. and Satterthwaite, M. (1983). Efficient Mechanisms for Bilateral Trading. *Journal of Economic Theory*, 29(2):265–281.

- Naor, M. and Stockmeyer, L. (1993). What can be computed locally? In *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 184–193, New York, NY, USA. ACM.
- Nassif, L. N., Nogueira, J. M., and Vinicius de Andrade, F. (2007). Distributed resource selection in grid using decision theory. In *CCGRID '07: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, pages 327–334, Washington, DC, USA. IEEE Computer Society.
- Navarro, G. and Borrell, J. (2006). An xml standards based authorization framework for mobile agents. In Burmester, M. and Yasinsac, A., editors, *Secure Mobile Ad-hoc Networks and Sensors*, volume 4074 of *Lecture Notes in Computer Science*, pages 54–66. Springer Berlin / Heidelberg. 10.1007/118014126.
- Negishi, T. (1962). The stability of a competitive economy: A survey article. *Econometrica: Journal of the Econometric Society*, pages 635–669.
- Nelson, M. (2009). Building an Open Cloud. *Science*, 324(5935):1656.
- Neumann, D., Stoesser, J., Anandasivam, A., and Borissov, N. (2007). Sorma—building an open grid market for grid resource allocation. *LECTURE NOTES IN COMPUTER SCIENCE*, 4685:194.
- Neumann, D., Stoesser, J., Weinhardt, C., and Nimis, J. (2008). A framework for commercial grids Economic and technical challenges. *Journal of Grid Computing*, 6(3):325–347.
- Neumann, D. G. (2007). *Market engineering : a structured design process for electronic markets*. PhD thesis, Karlsruhe.
- Nicolaisen, J., Petrov, V., and Tesfatsion, L. (2001). Market power and efficiency in a computational electricity market with discriminatory double-auction pricing. *IEEE Transactions on Evolutionary Computation*, 5(5):504–523.
- Nielsen, N. (1970). The allocation of computer resources pricing the answer? *Communications of the ACM*, 13(8):467–474.
- Nimis, J., Anandasivam, A., Borissov, N., Smith, G., Neumann, D., Wirström, N., Rosenberg, E., and Villa, M. (2008). Sorma — business cases for an open grid market: Concept and implementation. In *GECON '08: Proceedings of the 5th international workshop on Grid Economics and Business Models*, pages 173–184, Berlin, Heidelberg. Springer-Verlag.
- Nimis, J. et al. (2009). D2.2b: Final specification and design documentation of the sorma components revised version. Technical report, SORMA Consortium, EU-Project, <http://www.im.uni-karlsruhe.de/sorma/deliverables.htm>.
- Nisan, N. (2006). Chapter 9 Bidding Languages for Combinatorial Auctions . *Combinatorial auctions*, page 215.

- Niu, J., Mmoloke, A., McBurney, P., and Parsons, S. (2009). Catp: A communication protocol for cat games (version 2). Technical report, Technical Report, Department of Computer Science, Graduate School and University Center, City University of New York, New York, NY.
- Nurmi, D., Wolski, R., Grzegorzczak, C., Obertelli, G., Soman, S., Youseff, L., and Zagorodnov, D. (2009). The eucalyptus open-source cloud-computing system. *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 124–131.
- Nwana, H. S. (1996). Software agents: an overview. *The Knowledge Engineering Review*, 11(03):205–244.
- Ostermann, S., Iosup, A., Yigitbasi, N., Prodan, R., Fahringer, T., and Epema, D. (2010). A performance analysis of ec2 cloud computing services for scientific computing. In Akan, O., Bellavista, P., Cao, J., Dressler, F., Ferrari, D., Gerla, M., Kobayashi, H., Palazzo, S., Sahni, S., Shen, X. S., Stan, M., Xiaohua, J., Zomaya, A., Coulson, G., Avresky, D. R., Diaz, M., Bode, A., Ciciani, B., and Dekel, E., editors, *Cloud Computing*, volume 34 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 115–131. Springer Berlin Heidelberg. 10.1007/978-3-642-12636-9₉.
- Papadimitriou, C. and Tsitsiklis, J. (1987). The complexity of Markov decision processes. *Mathematics of operations research*, 12(3):441–450.
- Papazoglou, M., Traverso, P., Dustdar, S., and Leymann, F. (2007). Service-oriented computing: State of the art and research challenges. *Computer*, pages 38–45.
- Papazoglou, M. and van den Heuvel, W. (2007). Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, 16(3):389–415.
- Pardoe, D. and Stone, P. (2007). An autonomous agent for supply chain management. In Adomavicius, G. and Gupta, A., editors, *Handbooks in Information Systems Series: Business Computing*. Elsevier.
- Park, S., Durfee, E., and Birmingham, W. (1999). An adaptive agent bidding strategy based on stochastic modeling. *Proceedings of the third annual conference on Autonomous Agents*, pages 147–153.
- Park, S., Durfee, E., and Birmingham, W. (2000). Emergent properties of a market-based digital library with strategic agents. *Autonomous Agents and Multi-Agent Systems*, 3(1):33–51.
- Park, S., Durfee, E. H., and Birmingham, W. P. (1996). Advantages of strategic thinking in multiagent contracts (a mechanism and analysis). In *In Proceedings of the Second International Conference on Multiagent Systems (ICMAS-96)*, pages 259–266. MIT Press.

- Park, S., Durfee, E. H., and Birmingham, W. P. (2004). Use of markov chains to design an agent bidding strategy for continuous double auctions. *Journal of Artificial Intelligence Research (JAIR)*, 22:175–214.
- Parkes, D. (2008). Computational mechanism design. In *Lecture notes of tutorials presented at the 10th Conference on Theoretical Aspects of Rationality and Knowledge TARK-05, Singapore*.
- Parkes, D. C., Kalagnanam, J., and Eso, M. (2001). Achieving budget-balance with vickrey-based payment schemes in exchanges. In *Proceedings of the 17th international joint conference on Artificial intelligence - Volume 2*, pages 1161–1168, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- Parsons, S., R.-A. J. A. and Klein, M. (2009). Auctions and bidding: a guide for computer scientists. *Communications of the ACM*.
- Paschke, A., Dietrich, J., and Kuhla, K. (2005). A logic based sla management framework. In *Semantic Web and Policy Workshop (SWPW) at ISWC 2005*.
- Paurobally, S., Tamma, V., and Wooldrdige, M. (2007). A framework for web service negotiation. *ACM Trans. Auton. Adapt. Syst.*, 2(4):14.
- Phelps, S. (2007). Evolutionary mechanism design. *PhD Thesis*.
- Phelps, S., Marcinkiewicz, M., and Parsons, S. (2006). A novel method for automatic strategy acquisition in n-player non-zero-sum games. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 705–712, New York, NY, USA. ACM.
- Phelps, S., McBurney, P., and Parsons, S. (2010a). Evolutionary mechanism design: a review. *Autonomous Agents and Multi-Agent Systems*, 21(2):237–264.
- Phelps, S., McBurney, P., and Parsons, S. (2010b). A novel method for strategy acquisition and its application to a double-auction market game. *Trans. Sys. Man Cyber. Part B*, 40(3):668–674.
- Plummer, D., Bittman, T., Austin, T., Cearley, D., and Smith, D. (2008). Cloud computing: Defining and describing an emerging phenomenon. *Gartner, June*, 17.
- Porter, M. et al. (2001). Strategy and the Internet. *Harvard business review*, 79(3):62–79.
- Poslad, S., Laamanen, H., Malaka, R., Nick, A., Buckle, P., and Zipl, A. (2001). Crum-pet: creation of user-friendly mobile services personalised for tourism. *IEE Conference Publications*, 2001(CP477):28–32.
- Pueschel, T. and Neumann, D. (2009). Management of Cloud Infrastructures: Policy-Based Revenue Optimization. *International Conference on Information Systems*, Pueschel2009:178.

- Püschel, T., Borissov, N., Macías, M., Neumann, D., Guitart, J., and Torres, J. (2007). Economically enhanced resource management for internet service utilities. *Web Information Systems Engineering–WISE 2007*, pages 335–348.
- Quiroigco, S., Assis, P., Westerinen, A., Baskey, M., and Stokes, E. (2004). Toward a formal common information model ontology. In Bussler, C., Hong, S.-k., Jun, W., Kaschek, R., Kinshuk, Krishnaswamy, S., Loke, S., Oberle, D., Richards, D., Sharma, A., Sure, Y., and Thalheim, B., editors, *Web Information Systems WISE 2004 Workshops*, volume 3307 of *Lecture Notes in Computer Science*, pages 11–21. Springer Berlin / Heidelberg. 10.1007/978-3-540-30481-4₂.
- Rahwan, I., Sonenberg, L., Jennings, N., and McBurney, P. (2007). STRATUM: A methodology for designing heuristic agent negotiation strategies. *Applied Artificial Intelligence*, 21(6):489–527.
- Rappa, M. (2004). The utility business model and the future of computing services. *IBM Systems Journal*, 43(1):32–42.
- Reeves, D., Wellman, M., MacKie-Mason, J., and Osepayshvili, A. (2005). Exploring bidding strategies for market-based scheduling. *Decision Support Systems*, 39(1):67–85.
- Regev, O. and Nisan, N. (2000). The POPCORN market. Online markets for computational resources* 1. *Decision Support Systems*, 28(1-2):177–189.
- Rittinghouse, J. (2009). *Cloud Computing: Implementation, Management, and Security*. CRC Press.
- Robles, S., Borrell, J., Bigham, J., Tokarchuk, L., and Cuthbert, L. (2001). Design of a trust model for a secure multi-agent marketplace. *Proceedings of the fifth international conference on Autonomous Agents*, pages 77–78.
- Robu, V. and Poutre, J. L. (2009). Designing bidding strategies in sequential auctions for risk averse agents. *Multi-Agent and Grid Systems*, (to appear).
- Rosenberg, F., Curbera, F., Duftler, M. J., and Khalaf, R. (2008). Composing restful services and collaborative workflows: A lightweight approach. *IEEE Internet Computing*, 12(5):24–31.
- Rosenschein, J. and Zlotkin, G. (1994). *Rules of Encounter: Designing Conventions for Automated Negotiation Among Computers*. MIT Press.
- Ross, S. M. (2006). *Simulation*. Elsevier, Amsterdam, 4. ed. edition.
- Roth, A. E. and Erev, I. (1995). Learning in extensive-form games: Experimental data and simple dynamic models in the intermediate term. *Games and Economic Behavior*, 8(1):164 – 212.
- Rothkopf, M. (2007). Thirteen reasons why the Vickrey-Clarke-Groves process is not practical. *Operations Research*, 55(2):191–197.

- Rubinstein, A. (1985). A bargaining model with incomplete information about time preferences. *Econometrica*, 53(5):pp. 1151–1172.
- Saltelli, A., Ratto, M., Andres, T., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M., and Tarantola, S. (2008). *Global sensitivity analysis : the primer*. Wiley, Chichester, West Sussex [u.a.]. Includes bibliographical references and index.
- Sandholm, T. and Lai, K. (2007). A statistical approach to risk mitigation in computational markets. In *Proceedings of the 16th international symposium on High performance distributed computing*, page 96. ACM.
- Sandholm, T., Lai, K., Andrade, J., and Odeberg, J. (2006). Market-based resource allocation using price prediction in a high performance computing grid for scientific applications. In *HPDC06: Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing*, pages 132–143. Citeseer.
- Sandholm, T., Lai, K., and Clearwater, S. (2008). Admission Control in a Computational Market. *Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, pages 277–286.
- Satterthwaite, M. and Williams, S. (1989). Bilateral trade with the sealed bid k-double auction: Existence and efficiency. *Journal of Economic Theory*, 48(1):107–133.
- Schlieper, U. (1974). Tatonnement and non tatonnement processes of the price mechanism. *Journal of Economics*, 34(1):107–124.
- Schnizler, B. (2008). Mace: A multi-attribute combinatorial exchange. In Aalst, W., Mylopoulos, J., Sadeh, N. M., Shaw, M. J., Szyperski, C., Gimpel, H., Jennings, N. R., Kersten, G. E., Ockenfels, A., and Weinhardt, C., editors, *Negotiation, Auctions, and Market Engineering*, volume 2 of *Lecture Notes in Business Information Processing*, pages 84–100. Springer Berlin Heidelberg.
- Schwartzman, L. and Wellman, M. (2009). Stronger CDA strategies through empirical game-theoretic analysis and reinforcement learning. *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, 1:249–256.
- Sedgewick, R. and Wayne, K. (2010). *Algorithms*. Addison-Wesley Educational Publishers Inc, USA.
- Selten, R., Mitzkewitz, M., and Uhlich, G. R. (1997). Duopoly strategies programmed by experienced players. *Econometrica*, 65(3):517–555.
- Sherstov, A. and Stone, P. (2005). Three automated stock-trading agents: A comparative study. In Faratin, P. and Rodriguez-Aguilar, J., editors, *Agent Mediated Electronic Commerce VI: Theories for and Engineering of Distributed Mechanisms and Systems (AMEC 2004)*, volume 3435 of *Lecture Notes in Artificial Intelligence*, pages 173–187. Springer Verlag, Berlin.

- Shi, B., G.-E. H. V. P. and Jennings, N. R. (2010). A game-theoretic analysis of market selection strategies for competing double auction marketplaces. In: *9th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS2010)*, 14th-18th May, Toronto, Canada., pages 857–864.
- Shneidman, J., Ng, C., Parkes, D. C., AuYoung, A., Snoeren, A. C., Vahdat, A., and Chun, B. (2005). Why markets could (but don't currently) solve resource allocation problems in systems. In *HOTOS'05: Proceedings of the 10th conference on Hot Topics in Operating Systems*, pages 7–7, Berkeley, CA, USA. USENIX Association.
- Shoham, Y. (2008). Computer science and game theory. *Communications of the ACM*, 51(8):74–79.
- Shoham, Y. and Leyton-Brown, K. (2009). *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press New York, NY, USA.
- Shoham, Y., Powers, R., and Grenager, T. (2004). Multi-agent reinforcement learning: a critical survey. In *AAAI Fall Symposium on Artificial Multi-Agent Learning*, volume 3, pages 15–18. Citeseer.
- Shoham, Y., Powers, R., and Grenager, T. (2007). If multi-agent learning is the answer, what is the question? *Artificial Intelligence*, 171(7):365–377.
- Smirnova, O. (2009). Extended Resource Specification Language. [Online], <http://www.nordugrid.org/documents/xrsl.pdf>.
- Smith, R. (1980). The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on computers*, 29(12):1104–1113.
- Smith, V. (1982). Microeconomic systems as an experimental science. *The American Economic Review*, 72(5):923–955.
- Smith, W., Foster, I. T., and Taylor, V. E. (1998). Predicting application run times using historical information. In *IPPS/SPDP '98: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 122–142, London, UK. Springer-Verlag.
- Sodomka, E., Collins, J., and Gini, M. (2007). Efficient statistical methods for evaluating trading agent performance. In *AAAI'07: Proceedings of the 22nd national conference on Artificial intelligence*, pages 770–775. AAAI Press.
- SORMA (2008). D4.1:description of the security framework of the open grid market. Technical report, SORMA Consortium, EU-Project, <http://www.im.uni-karlsruhe.de/sorma/deliverables.htm>.
- Stavins, R. N., Wagner, A. F., and Wagner, G. (2003). Interpreting sustainability in economic terms: dynamic efficiency plus intergenerational equity. *Economics Letters*, 79(3):339 – 343.

- Stone, P. (2007a). Learning and multiagent reasoning for autonomous agents. In *The 20th International Joint Conference on Artificial Intelligence*, pages 13–30.
- Stone, P. (2007b). Multiagent learning is not the answer. it is the question. *Artificial Intelligence*, 171(7):402–405.
- Stone, P., Schapire, R., Littman, M., Csirik, J., and McAllester, D. (2003). Decision-theoretic bidding based on learned density models in simultaneous, interacting auctions. *Journal of Artificial Intelligence Research*, 19(1):209–242.
- Sturm, A. and Shehory, O. (2004). A framework for evaluating agent-oriented methodologies. In Giorgini, P., Henderson-Sellers, B., and Winikoff, M., editors, *Agent-Oriented Information Systems*, volume 3030 of *LNCS*, pages 94–109. Springer.
- Such, J., Alberola, J., Garcia-Fornes, A., Espinosa, A., and Botti, V. (2009). Kerberos-based secure multiagent platform. In Hindriks, K., Pokahr, A., and Sardina, S., editors, *Programming Multi-Agent Systems*, volume 5442 of *Lecture Notes in Computer Science*, pages 197–210. Springer Berlin / Heidelberg. 10.1007/978-3-642-03278-3_13.
- Sun, R. and Peterson, T. (1999). Multi-agent reinforcement learning: weighting and partitioning. *Neural Networks*, 12(4-5):727–753.
- Sutherland, I. (1968). A futures market in computer time. *Communications of the ACM*, 11(6):449–451.
- Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.
- Sycara, K. (1998). Multiagent systems. *AI magazine*, 19(2):79.
- Sycara, K., Widoff, S., Klusch, M., and Lu, J. (2002). Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. *Autonomous Agents and Multi-Agent Systems*, 5:173–203. 10.1023/A:1014897210525.
- Symantec (2008). State of the data center regional data global second annual report. Technical report, Symantec.
- Tamma, V., Phelps, S., Dickinson, I., and Wooldridge, M. (2005). Ontologies for supporting negotiation in e-commerce. *Engineering applications of artificial intelligence*, 18(2):223–236.
- Taylor, M. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research*, 10:1633–1685.
- Tesauro, G. (2007). Reinforcement Learning in Autonomic Computing: A Manifesto and Case Studies. *IEEE internet computing*, pages 22–30.
- Tesauro, G. and Bredin, J. (2002). Strategic sequential bidding in auctions using dynamic programming. In *Autonomous agents and multiagent systems*.
- Tesauro, G., Chess, D. M., Walsh, W. E., Das, R., Segal, A., Whalley, I., Kephart,

- J. O., and White, S. R. (2004). A multi-agent systems approach to autonomic computing. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 464–471, Washington, DC, USA. IEEE Computer Society.
- Tesauro, G. and Das, R. (2001). High-performance bidding agents for the continuous double auction. In *EC '01: Proceedings of the 3rd ACM conference on Electronic Commerce*, pages 206–209, New York, NY, USA. ACM.
- Tesfatsion, L. (2006). Agent-based computational economics: A constructive approach to economic theory. In Tesfatsion, L. and Judd, K. L., editors, *Handbook of Computational Economics*, volume 2 of *Handbook of Computational Economics*, chapter 16, pages 831–880. Elsevier.
- TheCloudMarket (2010). The cloud market, <http://thecloudmarket.com>.
- Thielscher, M. (2010). A general game description language for incomplete information games. In *AAAI*.
- Tran, Q. N. and Low, G. (2008). Mobmas: A methodology for ontology-based multi-agent systems development. *Information and Software Technology*, 50(7-8):697 – 722.
- Treadwell, J. (2007). Open Grid Services Architecture Glossary of Terms Version 1.6. Technical report, Open Grid Forum.
- Tsitsiklis, J. (1994). Asynchronous stochastic approximation and q-learning. *Machine Learning*, 16(3):185–202.
- Tsvetovatyy, M., Gini, M., Mobasher, B., Ski, Z., and Ski, W. (1997). Magma an agent based virtual market for electronic commerce. *Applied Artificial Intelligence*, 11(6):501–523.
- Turner, M., Budgen, D., and Brereton, P. (2003). Turning software into a service. *Computer*, pages 38–44.
- van Dinther, C. (2007). *Adaptive bidding in single-sided auctions under uncertainty : an agent-based approach in market engineering*. PhD thesis, Basel.
- van Ittersum, K., Pennings, J., Wansink, B., and Van Trijp, H. (2007). The validity of attribute-importance measurement: A review. *Journal of Business Research*, 60(11):1177–1190.
- van Valkenhoef, G., R.-S. V. P. J. N. and Verbrugge, R. (2009). Continuous double auctions with execution uncertainty. In: *Proc. IJCAI Workshop on Trading Agent Design and Analysis, Pasadena, USA.*, pages 113–122.
- Varian, H. R. (2009). *Intermediate Microeconomics: A Modern Approach*. W. W. Norton & Company; Eighth Edition edition.
- Vilajosana, X., Marques, J., Juan, A., and Krishnaswamy, R. (2009). A bidding spec-

- ification for Grid resources. *International Journal of Grid and Utility Computing*, 1(3):194–204.
- Vilajosana, X., Marques, J., Krishnaswamy, R., Juan, A., Amara-Hachmi, N., Navarro, L., and R&D, F. (2008). Bidding support for computational resources. *Proceedings of the 2008 International Conference on Complex, Intelligent and Software Intensive Systems-Volume 00*, pages 309–314.
- Von Neumann, J. and Morgenstern, O. (1944). *Theory of Games and Economic Behavior*. Princeton University Press.
- Vorobeychik, Y., Wellman, M., and Singh, S. (2007). Learning payoff functions in infinite games. *Machine Learning*, 67(1):145–168.
- Vrba, P. (2003). Java-based agent platform evaluation. In Mark, V., McFarlane, D., and Valckenaers, P., editors, *Holonc and Multi-Agent Systems for Manufacturing*, volume 2744 of *Lecture Notes in Computer Science*, pages 1086–1087. Springer Berlin / Heidelberg. 10.1007/978-3-540-45185-3₅.
- Vulkan, N. (2003). Automated trading in agent-based markets for communication bandwidth. *International Journal of Electronic Commerce*, 7(4):119–150.
- Vytelingum, P., Cliff, D., and Jennings, N. (2008). Strategic bidding in continuous double auctions. *Artificial Intelligence*, 172(14):1700–1729.
- Waldspurger, C., Hogg, T., Huberman, B., Kephart, J., and Stornetta, W. (1992). Spawn: A distributed computational economy. *IEEE Transactions on Software Engineering*, 18(2):117.
- Walsh, W., Tesauro, G., Kephart, J., and Das, R. (2004). Utility functions in autonomic systems. *Proceedings of the International Conference on Autonomic Computing*, pages 70–77.
- Watkins, C. and Dayan, P. (1992a). Q-learning. *Machine Learning*, 8(3):279–292.
- Watkins, C. and Dayan, P. (1992b). Technical note: Q-learning. *Machine learning*, 8(3):279–292.
- Weber, I., Barros, A., May, N., Hoffmann, J., and Kaczmarek, T. (2009). Composing services for third-party service delivery. In *ICWS '09: Proceedings of the 2009 IEEE International Conference on Web Services*, pages 823–830, Washington, DC, USA. IEEE Computer Society.
- Weinhardt, C., Anandasivam, A., Blau, B., Borissov, N., Meinl, T., Michalk, W., and Stösser, J. (2009). Cloud Computing—A Classification, Business Models, and Research Directions. *Business & Information Systems Engineering*, 1(5):391–399.
- Weiss, G. (2000). *Multiagent systems: a modern approach to distributed artificial intelligence*. The MIT press.

- Wellman, M. P. (2006). Methods for empirical game-theoretic analysis. In *AAAI'06: proceedings of the 21st national conference on Artificial intelligence*, pages 1552–1555. AAAI Press.
- Wellman, M. P., Greenwald, A., and Stone, P. (2007). *Autonomous Bidding Agents: Strategies and Lessons from the Trading Agent Competition*. MIT Press.
- Weske, M. (2007). *Business Process Management: Concepts, Languages, Architectures*. Springer-Verlag New York Inc.
- Whiteson, S. and Stone, P. (2006). Evolutionary Function Approximation for Reinforcement Learning. *The Journal of Machine Learning Research*, 7:877–917.
- Wilkes, J. (2008). Utility functions, prices, and negotiation. *Technical report, HP Laboratories*.
- Windsor, W., Rosenberg, E., and Villa, M. (2009). D6.1 – case definition and evaluation criteria. Technical report, SORMA Consortium, EU-Project, <http://www.im.uni-karlsruhe.de/sorma/deliverables.htm>.
- Wolski, R., Plank, J., Brevik, J., and Bryan, T. (2001). Analyzing market-based resource allocation strategies for the computational grid. *International Journal of High Performance Computing Applications*, 15(3):258.
- Wood, T., Gerber, A., Ramakrishnan, K., Shenoy, P., and Van der Merwe, J. (2009). The Case for Enterprise-Ready Virtual Private Clouds. In *Proceedings of the Usenix Workshop on Hot Topics in Cloud Computing (HotCloud)*, San Diego, CA.
- Wooldridge, M. and Jennings, N. R. (1995). Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10(02):115–152.
- Wurman, P. (2001). Dynamic pricing in the virtual marketplace. *Internet Computing, IEEE*, 5(2):36–42.
- Wurman, P., Wellman, M., and Walsh, W. (1998). The Michigan Internet AuctionBot: a configurable auction server for human and software agents. *Proceedings of the second international conference on Autonomous agents*, pages 301–308.
- Wurman, P., Wellman, M., and Walsh, W. (2001). A parameterization of the auction design space. *Games and Economic Behavior*, 35(1-2):304–338.
- Xu, H. and Shatz, S. (2003). Adk: An agent development kit based on a formal design model for multi-agent systems. *Automated Software Engineering*, 10:337–365. 10.1023/A:1025859021913.
- Yeo, C. and Buyya, R. (2006). A taxonomy of market-based resource management systems for utility-driven cluster computing. *Software Practice and Experience*, 36(13):1381.
- Yu, H. and Vahdat, A. (2006). The costs and limits of availability for replicated services. *ACM Transactions on Computer Systems (TOCS)*, 24(1):113.

-
- Zambonelli, F., Jennings, N. R., and Wooldridge, M. (2003). Developing multiagent systems: The gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12(3):317–370.

Nikolay Borissov
Seeguetstrasse 9
8804 Au ZH
Schweiz

Erklärung

(gemäß §4, Abs. 4 der Promotionsordnung vom 21.4.1989)

Ich versichere wahrheitsgemäß, die Dissertation bis auf die in der Abhandlung angegebene Hilfe selbständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und genau kenntlich gemacht zu haben, was aus Arbeiten anderer und aus eigenen Veröffentlichungen unverändert oder mit Abänderungen entnommen wurde.

Karlsruhe, 14.06.2011