Agile Market Engineering

Bridging the gap between business concepts and running markets

Zur Erlangung des akademischen Grades eines
Doktors der Wirtschaftswissenschaften

(Dr. rer. pol.)

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

genehmigte

DISSERTATION

von

Dipl.-Wi.-Ing Carsten A. Block

Tag der mündlichen Prüfung: 20. Juli 2010
Referent: Prof. Dr. Christof Weinhardt
Korreferent: Prof. Dr. Wolf Fichtner

2010 Karlsruhe

Acknowledgements

This work would have not been possible without the continuous support and encouragement of many different persons. First, I would like to thank my thesis advisor Prof. Dr. Christof Weinhardt for his untiring support, his visionary ideas, and for the freedom to develop and explore new ideas in the context of electronic markets. All this proved to be a very fruitful foundation for shaping my own research agenda.

Also, I would like to thank Prof. Dr. Wolf Fichtner for co-advising me and for providing valuable feedback on energy markets. Many thanks to my committee members Prof. Dr. Gerhard Satzger and Prof. Dr. Kay Mitusch for their support.

Much of the content of this work was inspired by countless discussions with my colleagues of the Institute of Information Systems at Karlsruhe Institute of Technology. Particular thanks deserve Prof. Dr. Dirk Neumann and Dr. Clemens van Dinther for the many fruitful discussions. Also, I would like to thank Dr. Henning Ahlert, Alexander Schuller, and Anders Dalen for sharing their knowledge about energy markets.

Parts of this book originate from work accomplished in MEREGIO, a smart grid project funded by the German Federal Ministry of Economics and Technology. The KIT consortium in this project is headed by Prof. Dr. Hartmut Schmeck and coordinated by Dr. Andreas Kamper. I would like to thank both as well as all other colleagues in this project for their continued support and for the constructive discussions throughout my time as project member.

I would like to express my gratitude to Dr. Wolf Ketter and Dr. John Collins for providing me with their expertise in the field electronic agents and for the joint development of the trading agent competition for energy markets. Two research stays in Montreal as guest member of Prof. Dr. Gregory Kersten's team helped me better understand electronic negotiations and electronic decision support. This knowledge proved to be very valuable while developing some of the software artifacts supporting the agile market engineering process model. Thank you so much!

I am deeply indebted to my family and my friends for their enduring patience and support throughout this long journey and I would like to dedicate this work to my wife Eva and my daughter Linnea-Marie. You are wonderful!

Karlsruhe, July 2010 Carsten A. Block

Abstract

Since the advent of the Internet, electronic commerce has become an integral part of everyday life in most parts of the world. In 2009 private consumers in Germany purchased goods and services worth 15.5bn EUR via e-commerce market places on the Internet, an increase of 14.4% over the year before. In this light, design and implementation of electronic market platforms becomes very important. Consequently, much scientific research has been devoted to the question of *"How to design electronic markets right?"* Mechanism Design, a specialized discipline in theoretical economics, focuses on exactly this question and tries to find optimal market rules for a given market environment. From this research stream many intriguing insights and results on optimal market design have evolved. Still most of these results are restricted by rather strong assumptions *"as economists tend to provide models with attractive equilibrium properties and recognizable strategic behaviors, often maintaining limiting assumptions to take the edge off of the computational difficulties. Computationally minded researchers, on the other hand, assume very simple player strategies to assure nice equilibrium behavior, allowing them to focus on the complex bidding and decision making environments"* (Anandalingam et al., 2005). Overall, explanatory power of these results for real markets is limited. By today's research standards market reality is simply too complex to be comprehensively covered and described by means of theoretic market models of one form or the other.

The agile market engineering process model developed as part of this thesis builds on this insight and assumes that electronic markets for the real world cannot be completely designed upfront due to complexity constraints and furthermore acknowledges that even small design changes can have a significant impact on user behavior and market outcome. Instead, the proposed process model tries to bridge the gap between theoretic market design and practical (electronic) market platform development.

Different to previous market engineering process models, it does not rely on extensive and detailed upfront design and modeling of a new electronic market platform. Instead it fosters and supports short, lightweight, incremental development cycles, each delivering a fully functional and incrementally improved market system that can be tested and evaluated "hands-on" as a running system. The rationale behind this is to pragmatically address the inherent "wickedness" and complexity of market design by relying on frequent feedback from market participants and thus by following

a flexible, iterative design-and-build strategy. Like this, a continuous improvement and change process is established that favors learning, pragmatic development, and refactoring over extensive upfront design and theoretical modeling.

Several software artifacts have been developed that are designed to support and facilitate agile market engineering projects in practice. The *Market Design Knowledge Base* is a software platform for storage and retrieval of market (mechanism) knowledge. Its parametric approach to describing market engineering knowledge is flexible enough to preserve insights on how (or how not) to design and realize electronic market platforms, optionally also covering information on the respective market environment, the traded products, or the types of market participants a specific market design recommendation is valid for. The *Market Repository* was built to support market developers in quickly instantiating new market instances from a set of previously developed and archived market platforms. Instead of developing new markets from scratch every time, and instead of using a generic but complex and hard to extend market runtime environment, small, simple, and specific *Market Templates* have been developed as technical foundations to built new electronic market platforms upon. Three basic market templates, one for a continuous double auction market, one for a call market, and one specific prediction market template were developed and published in the market repository as initial market templates. An *Experiment Center* was developed to provide a convenient to use, agent-based simulation environment as realistic test bed for newly developed market platforms. It allows for an extensive and realistic testing of market platforms in cases where market development teams are unsure about the fundamental market design. It consists of computer-grid enabled *Experiment Runners* for parallelization of simulation runs, a central simulation *Experiment Control Center*, and a *Time Series Data Store* for the unified provisioning of historic market data, which can be used to bootstrap electronic agents in preparation for simulations.

The agile market engineering process model and its accompanying software artifacts have been successfully applied in five different real-life market engineering projects. Results and insights from these case studies are reported and evaluated.

Contents

List of Figures

List of Tables

1 Introduction

"The hope that rational design by an omniscient planner could supersede practical knowledge derived from a process of adaptation and discovery swept across many fields in the course of the twentieth century. This approach was generally described as modernism."

John Kay, 2010

Since the advent of the Internet, electronic commerce has become an integral part of everyday life in most parts of the world. In 2009 private consumers in Germany purchased goods and services worth 15.5bn EUR via e-commerce market places on the internet, an increase of 14.4% over the year before (Kaldik and Eisenblaetter, 2010). In the same year about 14% of all sales in Germany were conducted via e-commerce platforms (Bauer and Czajka, 2009). In this light, design and implementation of electronic market platforms becomes very important. Consequently, much scientific research hast been devoted to the question of *"How to design electronic markets right?"* Mechanism Design, a specialized discipline in theoretical economics, focuses on exactly this question and tries to find optimal market rules for a given market environment. From this research stream many intriguing insights and results on optimal market design have evolved. Still most of these results are restricted by rather strong assumptions as *"economists tend to provide models with attractive equilibrium properties and recognizable strategic behaviors, often maintaining limiting assumptions to take the edge off of the computational difficulties. Computationally minded researchers, on the other hand, assume very simple player strategies to assure nice equilibrium behavior, allowing them to focus on the complex bidding and decision making environments."* (Anandalingam et al., 2005). Overall, explanatory power of the results for real markets is limited. By todays research standards market reality is simply too complex to be comprehensively covered and described by theoretic market models of one form or the other.

Rittel and Webber (1973) note that *"the search for scientific bases for confronting problems of social policy [e.g. allocation rules of markets] is bound to fail, because of the nature of these problems. They are 'wicked' problems, whereas science has developed to deal with 'tame' problems. Policy problems cannot be definitively described. Moreover, in a pluralistic society there is nothing like the undisputable public good;*

there is no objective definition of equity; policies that respond to social problems cannot be meaningfully correct or false; and it makes no sense to talk about 'optimal solutions' to social problems [e.g. welfare maximizing distribution of goods] unless severe qualifications are imposed first. Even worse, there are no 'solutions' in the sense of definitive and objective answers."

Kay (2010, p.97) argues that "wicked problems" are *"best tackled, not by moral algebra, but obliquely: they involve high-level objectives achieved through adaptation and iteration, with constant rebalancing of incompatible and incommensurable components that are imperfectly known but acquired as the process goes on. Honda understood the power of obliquity, McNamara's Ford did not. We see the results today"*.

Based on these insights, a new market engineering process model was developed as part of this thesis. It builds on the assumption that electronic markets for the real world cannot be completely designed upfront as a complete market design would be too complex and even small design changes can have a significant impact on user behavior and market outcome (Klemperer, 2002b).

The proposed process model pragmatically tries to bridge the gap between theoretic market design and practical (electronic) market platform development by unifying concepts from theoretical market modeling and business concept development with those from agile software engineering. Initially, high level economic market objectives such as *allocative efficiency* (see e.g. Gode and Sunder, 1993; Wurman, 1999) or *information efficiency* (see e.g. Verrecchia, 1979) are defined as desiderata for a newly developed market (Neumann, 2007). With these goals in mind the market development then takes place in small, incremental cycles each incorporating end user feedback that is based on "hands-on" experiences and elicited from the running market. This approach is called *agile market engineering.*

Technically, several *blueprint market platforms* have been developed, all of them built on top of a modern software infrastructure. Each of these platforms is a small and concise but still fully functional implementation of one specific market mechanism (e.g. a continuous double auction), which can be easily redesigned, adapted, or changed. As such, these blueprint platforms provide a convenient base infrastructure to build new market engineering projects upon.

Objectives of this work

Each new market engineering project is unique and requires different solutions for different market environments, products, and market participants. Thus, the following research objectives guided the development of the agile market engineering process model throughout the course of this thesis project.

1. The developed agile market engineering process model and all its supporting (software) artifacts should be designed as "building blocks" that *can* be used in conjunction with each other and in the process sequence proposed in this work. But there should be no necessity to do so and all of the artifacts should also be usable stand alone.

2. The whole agile market engineering process should be flexible and adjustable to the specific requirements of a project.
3. The market engineering process model should address the wickedness and complexity of electronic market design and implementation based on a continuous improvement process that is aimed at frequently delivering fully functional and incrementally improved electronic market platforms and that values user feedback from running markets more than extensive upfront market design.
4. A set of appropriate software artifacts should be developed that simplifies and technically supports the process of designing, implementing, testing and operating electronic market platforms.

Thesis Structure

This thesis is structured as follows. In Chapter 2 existing general purpose software engineering models, specific market engineering process models, and different market engineering software tool suites are reviewed. Based on this review, conceptual gaps in the existing models are identified and a set of requirements for an agile market engineering process model is developed in Chapter 3. Chapter 4 then describes the proposed agile market engineering process model as well several specific agile market engineering software artifacts that were developed to support agile market engineering projects in practice. In Chapter 5 several concrete market projects that were developed using the agile market engineering process model are described and evaluated. The thesis is concluded with a summary and outlook in Chapter 6.

Parts of this work have already been published before. van Dinther et al. (2006) describes a scenario for the development of future energy markets. It is refined and concretized into a roadmap in Block et al. (2008), which expects energy brokers to be established by approximately 2015, and estimates that power trading on an end consumer level will be possible by approximately 2018. Based in this roadmap Block (2007), Block et al. (2008), and Deindl et al. (2008) describe the idea of agent based energy market trading. A proof-of-concept market platform, described in Section 5.5, demonstrates the concept of market based scheduling for microgrids.

In Hirsch et al. (2010) the concept of online and offline energy market simulations based on historic data from a specific region is described. This ideas inspired the development of the TAC Energy competition described in Section 5.6. This section in turn is based work published in Block et al. (2010), Block et al. (2010a), and Block et al. (2010b). The principle idea of TAC Energy was first published in Block et al. (2009), which is currently in the second round of review for publication in the International Journal of Computer Aided Engineering. In Ahlert and Block (2010) an approach for generating artificial load and price forecasts is described, which is used within the TAC Energy project (see Section 5.6).

Insights from Neumann et al. (2007) and Block and Neumann (2008) served as the foundation for Section 4.2.1, which describes a knowledge base for storage and retrieval of market engineering knowledge. The description of the market engineering tool suite meet2trade described in Section 2.3.3 is mostly based on a publication by Kolitz et al. (2007).

In Schönfeld and Block (2010) the idea for a market template repository is described. This publication served as the basis for Section 4.2.2.

Part of the market simulation framework described in Section 4.2.3 is a data store for historic time series. As historic time series are sometimes difficult to acquire, several artificial time series generators were developed. One of them is an artificial driving profile generator for (electric) vehicles, which was jointly developed and described in Dietz et al. (2010).

The idea to apply browser automation technologies as technical basis for the execution of market interaction is described in Block and Chen (2007) and contributes to section Section 4.1.2.

2 Foundations & Related Work

With the rise of the Internet electronic commerce is quickly emerging. As of 2008 e-commerce sales constitute 39% of all sales in Germany (StatBA, 2009). For the same year Kaldik and Eisenblaetter (2010) calculated from a representative panel survey that private consumers purchased goods and services worth 13.3bn via e-commerce market places and 15.5bn EUR (14.4% increase) in 2009. In this light electronic marketplaces (EM) play an increasingly important role as key enablers of electronic commerce. Their main value propositions are (i) improved market efficiency through reduced transaction cost and (ii) increased market transparency (Lee and Clark, 1996; Garicano and Kaplan, 2001). An important question in this context is how to design electronic markets "right" so that they become successful and can actually "deliver" the transparency and market efficiency they promise.

Research dedicated to answering this question gained a lot of momentum over the last 15 years. Still, Wang et al. (2008) come to the conclusion that "*EM [Electronic Market] research field calls for more scientific researches. However, this effort could be dampened by the difficulty of collecting data [...] and a lack of specific EM directories or yellow pages.*" Anandalingam et al. (2005) find that scholarly research in this field so far mainly focuses on the economic fundamentals of electronic markets, i.e. the design of market mechanisms and auctions, and the understanding of buyer/seller behavior. They conclude that *"economists tend to provide models with attractive equilibrium properties and recognizable strategic behaviors, often maintaining limiting assumptions to take the edge off of the computational difficulties. Computationally minded researchers, on the other hand, assume very simple player strategies to assure nice equilibrium behavior, allowing them to focus on the complex bidding and decision making environments."* In other words market reality is seemingly too complex to be comprehensively covered and described by theoretic market models of one form or the other.

One of the reasons for the difficulty in modeling and describing electronic markets (and a general lesson learned from mechanism design research) is that details matter! Details pertain to the general market environment, to the properties of products and services traded in a market, to behavior, attitude and cultural background of market participants, and to the rules that govern markets. Only the combination of all of these factors constitutes a real market. Leaving out some of these aspects in simplified scientific market models significantly reduces the descriptive power of the results.

This is why some of the most powerful results from economic theory on market design are in the form of impossibility theorems such as the one of Myerson and Satterthwaite (1983) who prove that in a *Myerson-Satterthwaite environment* a market *can not* arrange efficient trade.[1]. Another result is the finding of Nisan and Ronen (2001) who show that certain classes of economic mechanisms (so called Vickrey-Clarke-Groves auction mechanisms) cannot be implemented in a computationally tractable form without sacrificing at least one of the other desired mechanism properties such as incentive compatibility or budget balance.

Insights from experimental economics on user behavior (e.g. Katok and Roth, 2004; Strecker and Seifert, 2003; Seifert and Ehrhart, 2005; Adam et al., 2008) and observations on trading behavior and market outcomes in real life markets (Klemperer, 2002a,b; Cramton, 2003; Campbell et al., 2005) add valuable insights to the understanding of market dynamics though some of the results are puzzling from a traditional economic point of view (Gimpel, 2006; Block et al., 2006).

A single best market mechanism or even a comprehensive framework that explains and prescribes particular market mechanisms for specific market environments does not exist. Moreover, Kolitz and Neumann (2007) and Kolitz (2008) find that not only economic mechanism design but also (software) system design can have significant influence on the performance and outcomes of electronic markets.[2] This finding adds another factor of uncertainty to the overall market engineering problem, which needs to be addressed during design and implementation of electronic markets.

To systematically support the complex process of creating new (electronic) markets from the initial vision, over a theoretical market concept, to the implementation and operation of an electronic market platform different (engineering) process models have been developed.

In their inner core electronic markets are – from a technical perspective – software systems. Consequently, the next sections review general purpose software development process models developed to provide systematic support and guidance

[1] In a *Myerson-Satterthwaite environment* buyer and seller of a good are assumed to know their own valuations for the good but not the one of the other counterpart. Furthermore, it is assumed that both valuations are independent from each other and that buyer and seller each know from which distribution the valuation of the other arises. Last but not least it is assumed that the support of the distributions overlap in order to rule out the trivial cases where the buyer always values the good higher than the seller so that trade could always happen – say – at a price equals to the valuation of the buyer, as well as the case where the valuation of the buyer is always lower than the one of the seller so that no trade will happen at all. In such an environment all equilibria of all possible mechanisms in which agents would voluntarily choose to participate are inefficient as compared to a setting with full information disclosure. For more details see e.g. McAfee et al. (1998)

[2] In lab experiments Kolitz et al. evaluated the performance of two different market mechanisms (a multi-attribute English auction and a multi-bilateral multi-attribute negotiation) implemented on two different technical market platforms (a web application vs. a client-server architecture) and found that not only the mechanism design but also the technical system design can significantly affect market outcome.

for developing software systems in general. Subsequently two specific process models dedicated to the development of electronic markets are investigated. Last not least special software artifacts that aim at supporting, shortening, and simplifying the development process for electronic markets are evaluated.

2.1 General Purpose Software Engineering Process Models

The IEEE computer society defines software engineering as *"(1) the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1)"* (IEEE, 1990). Building on this definition a software engineering process (model) is said to *"encompass the technical and managerial activities within the software life cycle processes that are performed during software acquisition, development, maintenance, and retirement"* (SWEBOK, 2004). In other words, such process models aim at providing guidelines on how to develop and operate information systems, with electronic market platforms being a specific subset of these. The following subsections provide an overview of several different process models and try to describe their specific advantages and disadvantages.

2.1.1 Classical Software Engineering Process Models

Many different software engineering process models have been proposed that all aim at streamlining the process of developing software. Among these models the **Waterfall Model** is one of the earliest. Its name stems from the corresponding process visualization that resembles a waterfall (see Figure 2.1) and was first described in literature by Royce (1970). In this model all required process steps follow one after

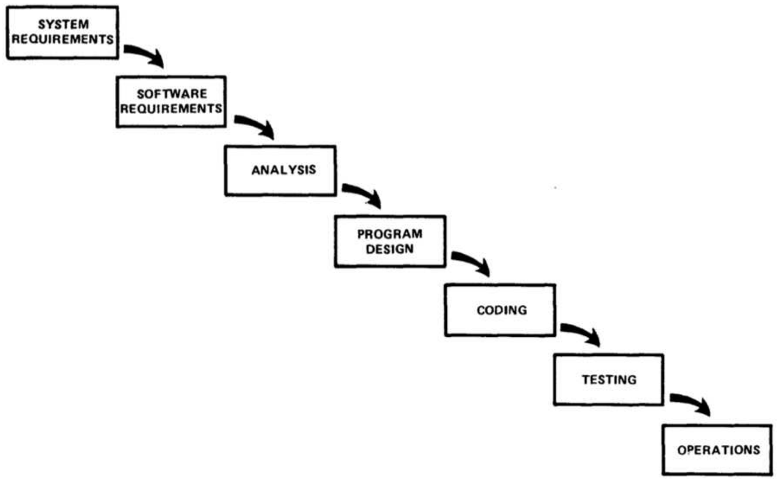


Fig. 2.1. The Waterfall Software Engineering Model (Source: Royce, 1970)

another beginning with a system requirements analysis and ending with a software's operations phase. The main idea is that one process step is accomplished at a time. Once, for example, the program design phase is finished it's results will have to be formally approved and then constitute the (fixed) input for the next phase. The rationale behind this idea is that a cautious and detailed software design at the beginning will save a lot of extra work in later process steps. McConnell (1996, p. 72) estimates that *"a requirements defect that is left undetected until construction or maintenance will cost 50 to 200 times as much to fix as it would have cost to fix at requirements time."*

Following this argument the **Big Design Upfront** model (commonly referred to as BDUF (Beck and Andres, 2004)) extends the idea of the Waterfall model in that it requires a complete and exhaustive design specification to be created and approved before any software implementation work is started.

The **V-Model** can be seen as a concrete extension of the two aforementioned models. Originally developed by German Federal Ministry of Defense it was revised twice (V-Model 97 (Dröschel and Wiemers, 2000) and V-Model XT (Broy and Rausch, 2005)) and is now the prescribed default software engineering model for all governmental software projects in Germany. Compared to the previously described models its main difference is that the V-Model differentiates process steps based on activities and results rather than on a timely basis. Also, a formal approval of process steps is not required. Figure 2.2 visualizes the process steps of the V-Model in the characteristic V-shape indicating increasing levels of detail the further down a process step is drawn on the "Vee".

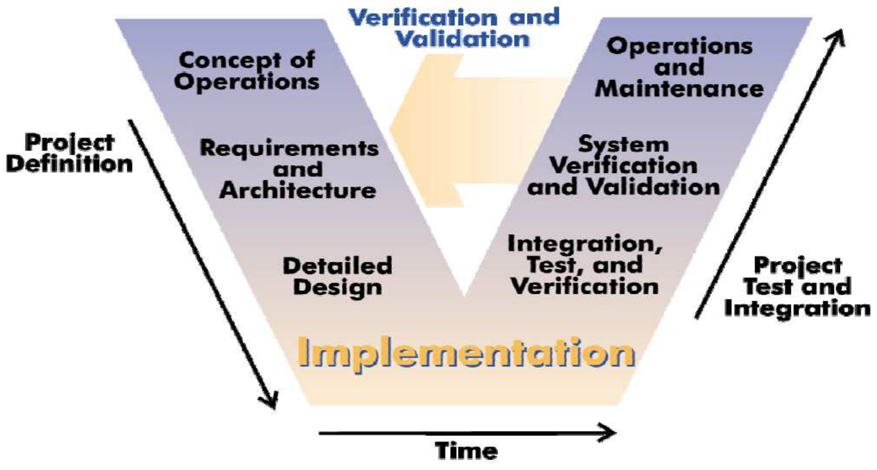

Verification and Validation
Concept of Operations
Requirements and Architecture
Detailed Design
Implementation
Integration, Test, and Verification
System Verification and Validation
Operations and Maintenance
Project Definition
Project Test and Integration
Time

Fig. 2.2. V-Model Process Visualization (Source: Osborne et al., 2005, p. 20)

Projects conducted according to the V-Model thus specify the project requirements in increasingly detailed form during the first process steps. After a detailed design concept is available implementation begins. Subsequently testing starts on a fine grained code level and subsequently becomes more holistic with integration, system verification and user acceptance tests. Each of the testing granularity levels thereby corresponds to a project specification detail level as indicated by the Vee shape. After testing is completed the system is rolled out and the operations and maintenance phase begins. Insights from this phase might then serve as input for follow-up projects.

Besides the aforementioned main advantage of avoiding errors early in the process, several points of critique are commonly raised when referring to these "traditional" types of software engineering process models:

- Life-Cycle process models (e.g. Waterfall model) lack flexibility to cope with requirement changes (McCracken and Jackson, 1982).
- Users are unable to specify their requirements upfront and learn what they want only during the process of creating the software. In other words: The system implementation changes the environment out which the need for the system arose, thus requirements will change too over time (DeGrace and Stahl, 1990).
- Life-Cycle process models assume that software engineers are able to foresee and solve potential implementation problems and challenges already during the design phases without any hands-on experiences from implementation. But even cautious initial design decisions that are subsequently implemented, tested but later prove to be wrong or erroneous are difficult and costly to correct in such a process model (Gladden, 1982).
- There exists a trade-off between extensive upfront design overhead and refactoring / error correction overhead afterwards. Initial design steps become useless if efforts expended on upfront modeling exceed those for refactoring the system afterwards. This problem is sometimes also referred to as *analysis paralysis* (Wiegers, 2000).
- Recent advancements in the area of software engineering tools and new, powerful, and concise programming languages in combination with sophisticated integrated development environments (IDEs) increase developer productivity significantly (Selby, 2007, pp. 170–171) and thus lower the cost for refactoring effectively tipping the trade-off balance between extensive upfront design and refactoring in favor of the latter one.
- Empirical research results *"indicate that smaller time frames, with delivery of software components early and often, will increase the success rate. Shorter time frames result in an iterative process of design, prototype, develop, test, and deploy small elements. This process is known as "growing" software, as opposed to the old concept of "developing" software."* (Standish Group, 1995) These results contradict extensive upfront planning in favor of short incremental release cycles.

Related to the last point of critique Larman and Basili (2003) note that *"by the late 1980s, the DoD [U.S. Department of Defense] was experiencing significant failure in acquiring software based on the strict, document-driven, single-pass waterfall model that DoD-Std-2167 required. A 1999 review of failure rates in a sample of earlier DoD projects drew grave conclusions: Of a total $37 billion for the sample set, 75% of the projects failed or were never used, and only 2% were used without extensive modification"*.

Around the same time an *agile software engineering movement* evolved[3] which intended to overcome traditional software engineering principles with all its shortcomings and to achieve a paradigm shift in this industry analogous to the paradigm shift that the *Toyota Production System* approach brought to traditional automative engineering, which – at that time – was still heavily inspired by traditional engineering ideals that rooted back to Henry Ford (Takeuchi and Nonaka, 1986).

2.1.2 Agile Software Engineering Process Models

Around the end of the 1990s an increasing number of software developers perceived traditional software engineering process models as too heavyweight, over-regulated, and prone to micromanagement behavioral patterns within project organizations (see Larman and Basili, 2003). Gloger (2009) claims that this type of *"industrialization"* of software engineering along the lines of mass production in (automotive) industry largely failed for the same reasons that traditional engineering in the automotive industry failed and eventually was superseded by ideas that root back to the Toyota Production System:

1. The (over-) application of division-of-labor principles to software engineering lead to the creation of jobs within the software engineering industry that were extremely narrow in scope (e.g. software test automation developer) and thus became boring or even devoid of meaning due to the extreme specialization.
2. Extensive division and distribution of tasks lead to the problem that nobody in the value creation chain felt responsible for the quality of the end product anymore. For traditional mass production Womack et al. (1991) estimates cost for "rework" at the end of the conveyor belts to be as high as 25% of the overall production cost.

Similar problems evolved in software engineering (see also Section 2.1.1) where bug fixing (the "rework" equivalent) took more and more time as job specialization in software engineering increased further. This finally lead to a situation in the late 1980s and early 19990s in which the majority of all software projects in the U.S. were cancelled or finished only after major rework Larman and Basili (2003).

In this situation a new *agile software engineering* movement gained a lot of momentum (Abrahamsson et al., 2009). Proponents of this new approach to software engineering advocate the use of short iterative software development cycles (each lasting only a few weeks), generally emphasize and promote the use of "light" and people-centric methods, and use frequent feedback loops as primary control mechanisms instead of extensive upfront planning. Poppendieck and Poppendieck (2006) propagate seven fundamental principles for agile software engineering:

Eliminate Waste: Waste in software engineering pertains to partially done work and unnecessary extra features. Specifying requirements long before coding likely produces overhead (churn) from frequent requirement changes. Likewise doing software testing long after the coding itself took place results in significant test-and-fix overhead. Both together are likely to lead to significant extra work from

[3] See `http://agilemanifesto.org`

extensive upfront specifications and late (sometimes called big-bang) integration. Consequently only the most value-adding features should be realized with least amount of code possible in short concept-implement-test-release cycles.

Build Quality In: Shingo (1989) distinguishes two forms of quality inspection, (i) inspection after the occurrence of defects and (ii) inspections to prevent defects. Achieving great quality requires to set up an environment that actively prevents defects ex-ante rather than one that reactively fixes them ex-post. For software engineering, test driven development (Beck, 2002) proves to be extremely valuable as an institutionalized way to prevent defects upfront.

Create Knowledge: Poppendieck and Poppendieck (2006) claim that software engineering is a knowledge-creating process and that *"the puzzling aspect of 'waterfall' development is the idea that knowledge, in the form of 'requirements', exists prior to and separate from coding."* Furthermore they note that an early design can neither anticipate the complexities discovered during the development phase nor can it take insights and new knowledge learned from and during implementation and usage of the software into account.

Defer Commitment: Irreversible decisions should be deferred as long as this is reasonably possible without causing damage. The rationale for this principle is that the later a decision is made the more knowledge is available supporting a qualified decision making. In this light, also the amount of irreversible decisions should be reduced as much as possible e.g. by using software development tools that ease the burden of continuous refactoring. Also, when faced with significant uncertainty about the likely future impact of a decision, several different options should be tried and tested.

Deliver Fast: Development should be accomplished in short iterations (a few months at most) with tasks to accomplish being small enough to not clogging the development process for a long time. Also the scheduling of future work should be limited to one or two iterations ahead of time as longer planning horizons are more prone to change and thus longer-term working plans represent avoidable and unnecessary extra efforts.

Respect People: A passionate team that is proud on is own craftsmanship and the results it delivers is invaluable. Trust and respect are fundamental pre-conditions for teams to become passionate. Another supporting factor is to move responsibilities and decision-making to the lowest possible level in the organization's hierarchy, i.e. to the developers themselves. In good parts respect can be expressed as trust in a team's skill to self-organize its work so that it reaches its goals.

Optimize the Whole: "Optimization" in the context of software development has two meanings. The first is devoted to the optimization of the resulting software itself. Here the only thing that really matters is return on invest: The software engineering process should be shaped such that features (requirements) with the highest value add (from an end customer perspective) are delivered first. The second meaning of "optimization" is dedicated to the development process itself. Frequent self-reflection – at best at the end of each single process iteration – helps to continuously optimize the development process.

In the following sections, three agile software engineering process models are described in more detail that build on the aforementioned principles. *Extreme Programming* is often referred to as an agile software *engineering* model. *Scrum* is often referred to as an agile software *management* model, and the *Crystal Clear Family* is often referred to as a set of lightweight building blocks for agile software development processes that allow for individual combination and thus a customization of the development procces.

Extreme Programming

Extreme Programming (XP) was originally invented by Kent Beck during a long-term industry project, which aimed at completely rewriting the payroll application of a large automotive company (Copeland, 2001). XP is considered a lightweight process model in that it discourages many of the heavy-weight principles of traditional software engineering such as extensive upfront requirements engineering or lengthy project documentation and instead promotes ideas such as keeping development teams small, synchronous and face-to-face communication processes, simple code, and – most importantly – short release cycles (Beck, 1999). Figure 2.3 visualizes the different timing of Waterfall and XP style software development.

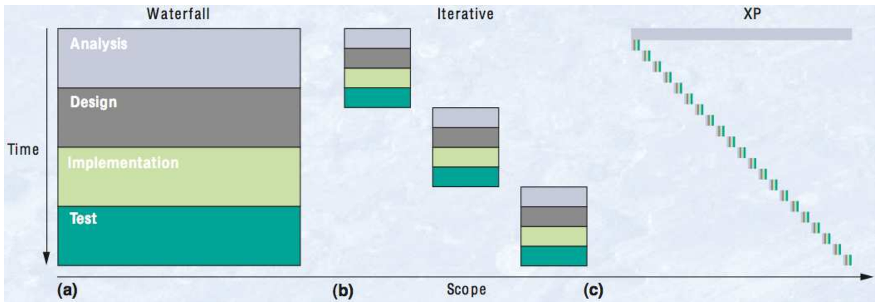


Fig. 2.3. Xtreme Programming as opposed to Waterfall Model (Source: Beck, 1999, p.70)

In XP a project starts with future end users (i.e. *on-site customers*) of the system writing short *user stories*. These are short, natural language texts that replace traditional (large) functional requirements documents and formulate expectations towards the future system from a customer point of view. Once an initial set of stories is collected the efforts for implementing each of the stories are estimated by the programmers (*planning game*) and then only this set of user stories is implement in the next development cycle. Then, instead of start writing the production software code itself, unit and functional *test code is written first* that basically formulates expectations on how the final production code should behave. With this testing harness in place the production code is then written in *pair programming* style, i.e. by two software developers sharing one keyboard, one mouse, and one monitor. The code is supposed to be written in the *simplest possible form* and without any repetitions such that all tests (written before) can be successfully executed without any errors. Code is always checked into code versioning systems so that all development team members have always access to the latest versions of all code fragments. Furthermore a *continuous software integration* tool continuously observes the versioning system, and automatically executes all necessary test and build operations after a change in the repository is detected providing developers with an immediate feedback on the quality of code checked in. If one or more tests in such an automated build fail, previously checked in changes are discarded. Like this, there is always a running system available built on the latest stable codebase of the project. Every few days or weeks a *small release* is made based on the latest stable code available. The customer then evaluates this release "hands-on" and – with this experience in mind – starts writing new user stories as input for the next process cycle. In the next iteration of the process the codebase is *refactored* such that it fulfills the new (possibly changed) customer requirements. These cycles continue to take place until the customer considers the software fully functional and no new or existing requirements are added or changed.

Scrum

While XP is strongly focused on the technicalities of software development, **Scrum** puts agile software *management* principles into its core focus. Ken Schwaber calls it a *"process skeleton on which hang all of its practices."* (Schwaber, 2009). As shown in Figure 2.4 work in Scrum is structured into short, fixed (timeboxed) cycles that are called *Sprints* and that last between two and four weeks.

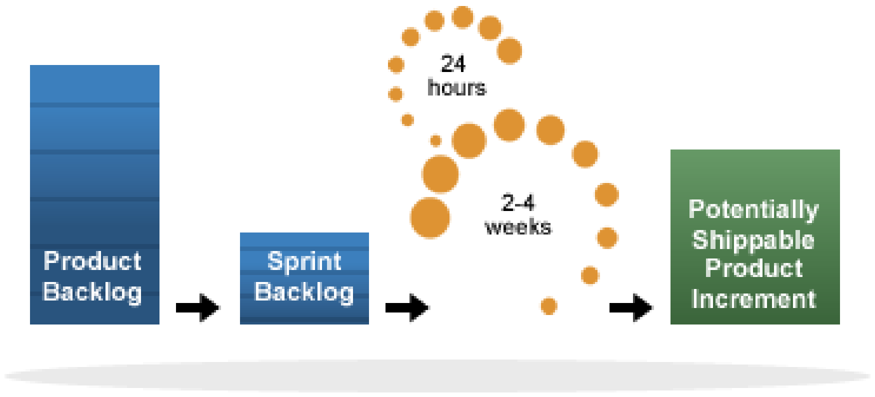


Fig. 2.4. The Scrum Process (Source: ScrumAlliance, 2010)

Similar to XP, in each sprint the development team takes a set of prioritized user stories from the product backlog and subsequently implements these stories during the following sprint such that at its end a feature-improved *potentially shippable* version of the software is delivered.

Scrum relies on three main technical artifacts, Product Backlog, Sprint Backlog and Burndown Chart.

- The *Product Backlog* contains all user stories written and prioritized by business value from the product owner.

- The *Sprint Backlog* contains a subset of the product backlog, namely those of the top most important user stories that the team thinks can be implemented within the fixed timeframe of the next sprint.
- During a sprint the *Burndown Chart* shows the cumulative remaining work measured in story points (see Section 4.1.2) over time, which still has to be accomplished before all stories scheduled for the respective sprint are completely implemented, tested and potentially shippable. Figure 2.5 shows a sample Burndown chart.

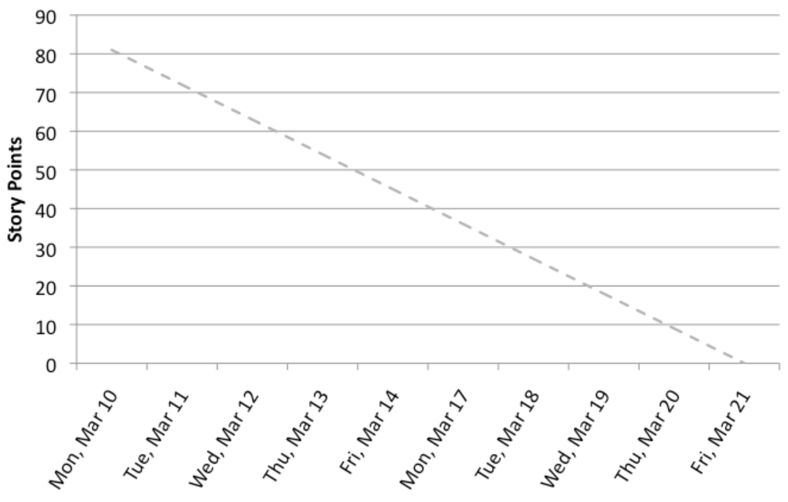


Fig. 2.5. Sketch of a Burndown Chart

Organizationally Scrum defines three main roles, *Scrum Master*, *Product Owner*, and *Team*:

The Scrum Master role is that of a facilitator who continually ensures that core scrum principles (such as strict time boxing of meetings and sprints) are obeyed. Also he tries to improve the efficiency of the team by identifying and addressing all types of personal, organizational and technical obstacles he gets to notice and by shielding the team from distractions during sprints. In that sense the Scrum Master ensures that the team can remain focused on its original task of developing software that realizes and fulfills the respective user stories.

The Product Owner represents the customer's interests (either as the customer himself or as a proxy person such as a key account manager). He is responsible for (i) maintaing a list of user requirements (user stories) in the product backlog and (ii) prioritizing them according to value-add (business value). As such the product owner role is there to ensure that the team works into the right direction from a business point of view and thus always implements the most important features first.

The team is a completely self-organizing entity and responsible for developing the product. Within each sprint the team designs, develops, and tests the currently selected user stories applying many of the aforementioned XP development principles such as continuous integrations and test-driven development as “good” software engineering practices. A team usually consists of 3 – 7 persons and should be composed such that it encompasses all skills required to realizing the user stories.

Three main recurring meetings are core to Scrum to organize the software development process: *Sprint Planning Meetings*, *Daily Scrum Meetings* and *Sprint Reviews*.

During the *Sprint Planning Meeting* product owner and team review the product backlog and discuss about the user stories so that the team can gain a detailed understanding of the requirements on each of the user stories. With this knowledge the team then estimates the required efforts for implementing each of the user stories. Then it pulls exactly that number of the upmost user stories from the product backlog (that was prioritized by the product owner before) into the sprint backlog that it (not the product owner) think can be realized within the next sprint. After the planning is finished the sprint begins.

At the end of each day a fifteen minute *daily scrum meeting* is held that is moderated by the Scrum Master. During this meeting each team member reports on (i) what he accomplished today, (ii) what he wants to accomplish tomorrow, and (iii) what obstacles refrained him from accomplishing his work. This meeting is meant to (i) retrieve a quick status report, facilitate team commitment, and (iii) provide a systematic opportunity to quickly react to occurring problems and to adapt work schedules accordingly.

At the end of a sprint the *Sprint Review* meeting takes place at which the team – guided by the Scrum Master – reflects on how the development process itself could be improved so that not only the software but also the software engineering process itself are subject to continuous improvement efforts.

More detailed descriptions on Scrum are provided, for example, by Schwaber and Beedle (2001); Gloger (2008); Wirdemann (2009).

Crystal Light

Crystal Light is a set of software engineering process models developed by Cockburn (2005). Crystal Light is different from XP and Scrum in that it does not prescribe a single process model for all projects but merely provides as set of core building blocks that can be combined and adjusted according to particular project needs. Additionally several concrete Crystal Light process models exist that can be seen as specially selected subsets of the Crystal Light building blocks. These models are named by colors and designed to be ready-to-use for projects of different criticality and different size. Larger projects that require more coordination and communication match to darker colors as shown in Figure 2.6. Crystal Clear (Cockburn, 2004) is the most lightweight member of the family and designed to fit the needs of small teams in (at most) modestly critical application environments.[4]

[4] “Critical” application contexts in the understanding of Cockburn include aircraft control software, software for nuclear power plant operations systems and the like.

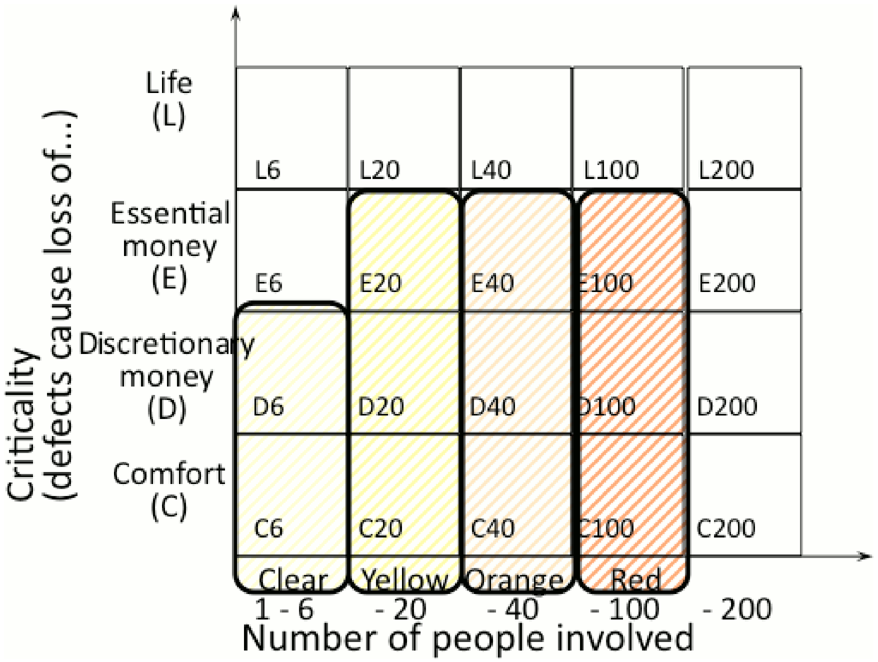


Fig. 2.6. Crystal Family Software Development Methods Overview (Source: Cockburn, 2005)

According to Cockburn (2004) seven base principles or core building blocks of agile software management exist that all aim at shifting a project as far as possible into what he calls s *"safety zone"* where project success becomes more likely. Combining these principles in one way or another leads to the differently colored Crystal models.

1. *Frequent Delivery* means that tested and running code is delivered to customers in short release cycles. Cockburn defines short as "less than four months". Frequent delivery ensures that customer feedback from a running system can be included early and often and helps users as well as developers to continuously learn and improve the software during development.

2. *Reflective Improvement* is not focusing on the software itself but on the underlying software development process. It means that teams should come together from time to time to reflect on how they can improve the development process by means of adjusting and tuning the way they work themselves such that they become more efficient and effective and their work is better harmonized with the one of others. Cockburn recommends to hold such meetings "frequently".

3. *Close Communication* (also called osmotic communication) is ensured by locating the development team in close vicinity to each other, at best in the same room. Like this, team members can get their questions answered within 30 seconds or less without leaving the room or using other means of communication. According to Cockburn in such an environment questions and answers simply "flow" with almost no disturbance on the team. Also information distributes

among team members functions more or less automatically with no or little written documentation required.

4. *Personal Safety* is the basic for trust among team members. Personal safety means that people can articulate their thoughts without fearing repressions. With personal safety in place problems and difficulties are likely to be articulated early and frankly so that they can be solved quickly. Also, team members might be more willing to ask for help, and to reveal their problems in solving an assigned task to others. Costa (2003) shows empirically that this type of trust is *"positively related with perceived task performance and with team satisfaction"*.

5. *Focus* means that the members in the development team (i) know what to work on, and (ii) have time and peace of mind to actually accomplish the work. The "what" is determined by business value, i.e. the – from a customer's point view – most important features of the software that should be developed first. These are best estimated by customers themselves or at least by domain experts and *not* by the developers of the software themselves. Once priorities are clear developers should be assigned one or at most two tasks at a time and be allowed for sufficient time to accomplish the work. With more than two tasks to accomplish in parallel and with insufficient time for the completion of the work, productivity as well as product quality both decrease drastically.

6. *Easy Access to Expert Users* (customers or domain experts) provides the development team with rapid feedback on design and quality of their software and also allows them to keep requirements up-to-date, or to adjust them where necessary. Keil and Carmel (1995) report that *"in 11 of the 14 paired cases, the more successful project involved a greater number of links [between customers and developers] than the less successful project. [...] This difference was found to be statistically significant in a paired t-test (p ¡ 0.01)."* According to Cockburn (2004) expert access is usually implemented by (i) weekly or bi-weekly customer-developer meetings, (ii) experienced users (customers) being directly part of the development team, and / or (iii) developers becoming trainee users for some time.

7. *Automated Testing & Integration* eases and automates some of the most tedious but nevertheless very important tasks of software development. It should be complemented by automated test coverage reporting tools, and automated configuration and dependency management solutions. Automated testing tools run unit- and integration tests upon each change in the code base providing instant feedback to developers on how their changes affect the overall system. Test coverage reports show how much of the production code is actually covered by unit- or integration tests pointing to classes and methods that are not yet verified by tests. Last but not least, tools for automated configuration and dependency management in combination with software that automatically integrates and builds releases from source code, help to bring down the cost for repeatedly creating binary snapshots of the software close to zero. As a result the development team will be able to provide frequent (e.g. nightly) builds of running and tested software – the technical foundation to the aforementioned principle of *frequent delivery.*

In summary agile software engineering process models such as the three described above claim to be more flexible in coping with requirement changes during project runtime, and thus to reduce documentation and reporting overhead significantly so that overall productivity increases. Empirical studies also report that the application of agile software engineering practices lead to a higher satisfaction for customers, developers and management, better code quality and shorter overall development times (Dyba and Dingsoyr, 2008).

However, critics claim claim that (i) agile software engineering process models are suitable only for small teams (Cohen et al., 2004), (ii) agile models work best for experienced teams where other models would have performed well too (Dyba and Dingsoyr, 2008), (iii) a lack of attention is put to design and architectural issues (McBreen, 2002), (iv) permanent pair programming and customer on-site rules of extreme programming are simply unrealistic and sometimes not even desirable (Stephens and Rosenberg, 2003), and (v) no long term experiences exist particularly with respect to maintenance and revisioning of software systems in a longer term perspective where minimal code documentation might lead to negative results.

Still, according to Forrester / Dr. Dobbs Global Developer Technographics Survey, Q3 2009 among 1298 IT professionals, *"35% of the respondents stated that 'Agile' most closely describes their development process"* (West and Grant, 2010, p. 2). Figure 2.7 shows that 30.6% of the respondents report that no formal process methodology is used at all in their company, 21% reportedly use some form of iterative model and only 13% build their software development processes upon different flavors of the Waterfall model. As of today, well established communities exist that continue

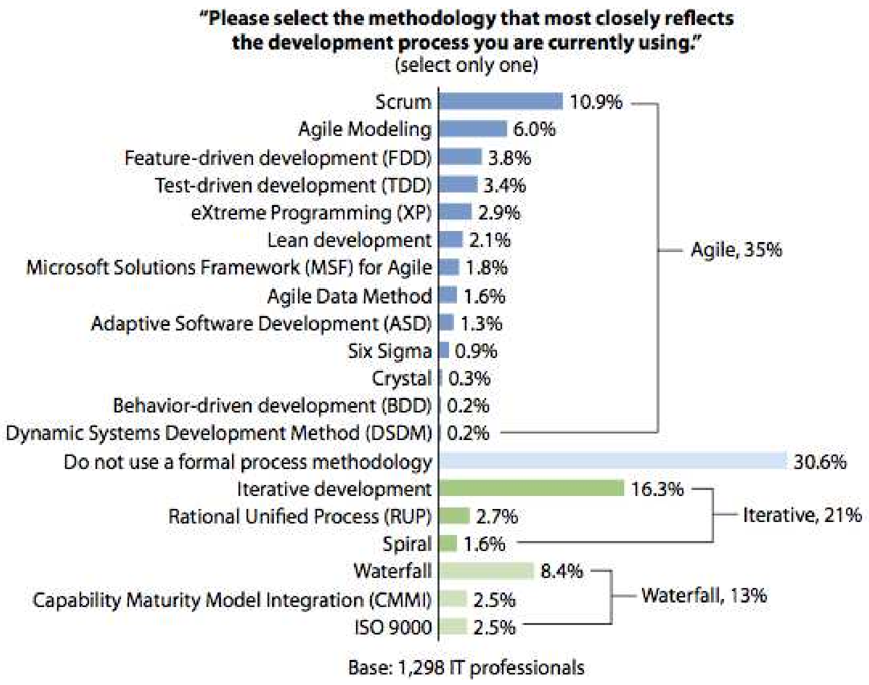


Fig. 2.7. Adoption of Agile Practices (Source: West and Grant, 2010))

to develop and refine agile software engineering process models, and that provide trainings and professional services for newcomers to easily adopt these methods.

While “Agile” ideas and methodologies have become a core part of modern software engineering the world looks different when it comes to specific market engineering process models. The following subsections review the current state of two of these specialized models.

2.2 Specific Market Engineering Process Models

In this section two specific market engineering process models are described in more detail. The first one was developed by Kambil and van Heck around the year 2002, the second one was mainly developed by Neumann and Weinhardt in the time between 2000 and 2006.

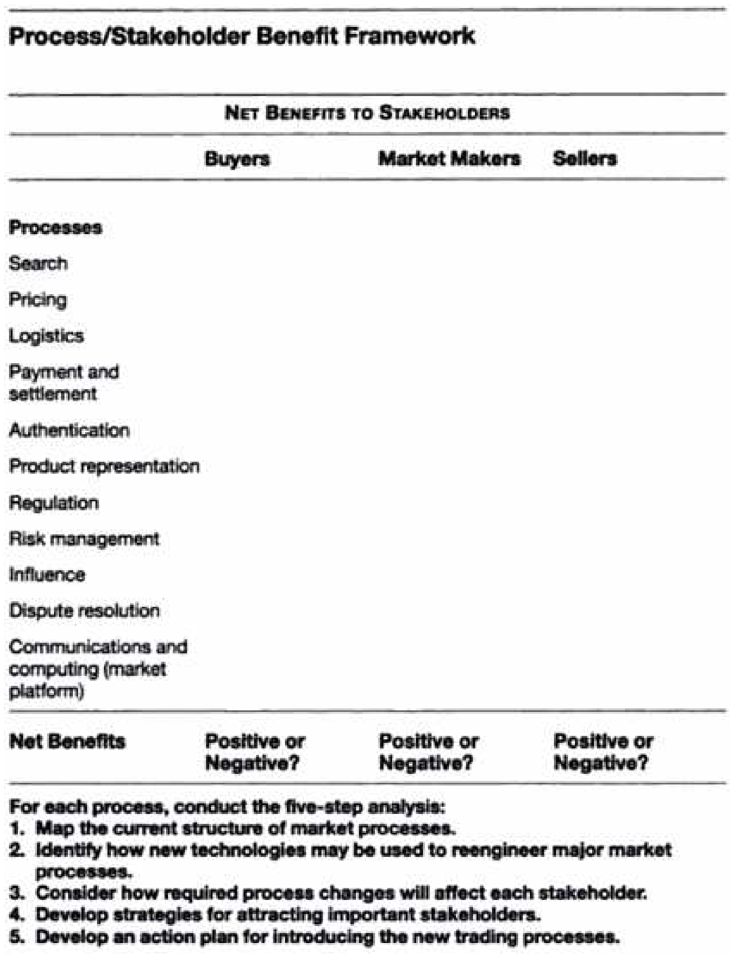
2.2.1 Process / Stakeholder Benefit Framework

Based on more than 100 industry use case studies Kambil and van Heck (2002) developed a set of recommendations on how to design electronic markets for different application scenarios in different industries. Based on this empirical evidence they also developed a market development process model called *“Process / stakeholder benefit framework”* that managers *“can use to systematically evaluate market-making initiatives”* (Kambil and van Heck, 2002, p. 64). This process model is mainly intended to (i) support managers in identifying optimization potential in existing markets, (ii) help them develop innovative solutions for identified shortcomings, and (iii) guide them on how to strategically introduce their innovation into the market by specifically identifying and targeting key customers and other beneficiaries. The main application context for the model is the introduction of electronic markets, in particular electronic auction markets, in various different business domains as well as the adaptation of established market processes *“from place to space”* (Kambil and van Heck, 2002, p. 21), i.e. the transformation of physically operating marketplaces into electronic counterparts.

Figure 2.8 shows the so called *“stakeholder benefit matrix”*, the core artifact of the model. Kambil and van Heck developed it to help managers identify technology opportunities by focusing on the vertical dimension of the matrix, which makes it easy to compare different market processes and their interdependencies amongst each other. Additionally, they point out that managers are also supported in identifying net benefits for the different market stakeholders by focusing on the horizontal dimension of the matrix.

For the practical application of the framework the authors developed the following five-step process model:

1. *State-of-the-Art Market Analysis:* In this step the various market related processes listed in Figure 2.8 need to be mapped and described for buyers and



Process/Stakeholder Benefit Framework

	Net Benefits to Stakeholders		
	Buyers	**Market Makers**	**Sellers**
Processes			
Search			
Pricing			
Logistics			
Payment and settlement			
Authentication			
Product representation			
Regulation			
Risk management			
Influence			
Dispute resolution			
Communications and computing (market platform)			
Net Benefits	**Positive or Negative?**	**Positive or Negative?**	**Positive or Negative?**

For each process, conduct the five-step analysis:
1. **Map the current structure of market processes.**
2. **Identify how new technologies may be used to reengineer major market processes.**
3. **Consider how required process changes will affect each stakeholder.**
4. **Develop strategies for attracting important stakeholders.**
5. **Develop an action plan for introducing the new trading processes.**

Fig. 2.8. Process / Stakeholder Benefit Framework (Kambil and van Heck, 2002, p. 65)

sellers of the particular market domain. This description of the status quo is then used as a baseline for further analysis. [5]

2. *Identification of business opportunity induced by new technology:* This step is focused on finding out if new technology can be used to reorganize and improve important established market processes like e.g. a logistics process. For each of the market processes listed in Figure 2.8 the authors provide several "blueprint questions" that are intended to give managers who apply the process model a feeling on the type of questions they need to ask. For the logistics process these blueprint questions include *"How can transition from place to space reduce logistics, distribution, and handling costs? Can the digital representation of products and the decoupling of logistics and pricing processes for trading create substantial new value? Who benefits from the gains and who incurs new cost?"* (Kambil and van Heck, 2002, p. 66). The authors also underline that this critical assessment

[5] Particular guidance on how this step could be accomplished best is not provided.

should not only comprise core market processes like allocation, pricing and settlement but also context processes like product representation, regulation, risk management, or dispute resolution.

3. *Estimation of stakeholder reactions:* Buyers, Sellers, and market makers may react differently to change happening in their market. In this process step the potential impact of the the particular change needs to be estimated. The main question is if the desired process change(s) create benefits beyond the ones in the existing model. Furthermore the distribution of these benefits is important. If e.g. buyers are expected to systematically gain value while sellers would systematically loose it is unlikely that the introduction of the new process will be successful. Kambil and van Heck also underline that changes in required workforce skills or in transaction cost for using the market have to be closely examined on their overall impact. The same applies to interdependencies of the newly introduced market process with other existing processes. In one example the authors point to a situation where the switching to digital product representation in an auction lead to significant cost savings but also made a revision of the closely coupled buyer authentication process necessary. Only if the estimated overall net benefits across all interdependent processes can be expected to be positive, the process innovation (e.g. a new electronic payment and settlement system) should be introduced into the market.

4. *Strategy definition for winning important stakeholders:* In this step a strategy has to be developed on how to get the key stakeholders, important buyers or sellers for example, to adopt the new process. According to the authors, possible options to achieve this goal include offering co-ownership to key stakeholders or to systematically subsidize early adopters of the new process as well as disadvantaged stakeholders. The benefit analysis from step three is supposed to provide helpful insight as it aims at identifying those stakeholders who benefit the least from the new process and who will consequently be reluctant to adopt the proposed new processes.

5. *Development and implementation of an action plan:* With the previous four steps accomplished, an action plan for the introduction of the new market process has to be developed. In this step questions on when to transform which market (sub) process have to be addressed and a concrete roadmap for the implementation of the required (software) artifacts needs to be established. Further details on how to develop and implement such an action plan are not provided, though a case study is quoted where a live auction was transformed into an online auction by Tele Flora Holland. In that particular action plan the managers of the firm decided to retain the well known auction format (i.e. the market rules) while changing the auction environment (from a live hall into a televised version of it). According to Kambil and van Heck the decision *not* to change the auction format ensured support from stakeholders and smoothed the transformation process as market participants were already familiar with the market rules.

Overall the Process / Stakeholder Benefit Framework is strongly focused on finding business opportunities in existing markets and on helping managers to develop new market ideas as well as strategic concepts for market entry. Detailed instructions on how to execute the different steps of the process model are not provided. Still a remarkable value-add comes from the many case study descriptions that Kambil and van Heck collected in order to provide managers with a wide range of practical insights and lessons learned on how markets or market processes are developed and adapted in practice.

2.2.2 The *Market Engineering* Process Model

Weinhardt et al. (2003); Neumann (2007) propose a structured and theoretically founded *"market engineering"* approach for analyzing, designing, implementing and quality assuring electronic market platforms. In other words their market engineering process model tries to bridge the gap between analytical (game theoretic) and experimental market design as frequently used in economics research (Smith, 1982) and classical software engineering based on traditional process models such as the Waterfall model described above.

In its core market engineering divides between an exogenously given and unchangeable *economic environment* (jurisdiction, knowledge, preferences, etc.) and a set of cautiously chosen *institutions*. Institutions are basically market rules that can be explicitly designed and adjusted by a market engineer. Together with the economic environment, institutions influence (incentivize) the *behavior* of the market participants without completely prescribing or dictating it. The behavior, i.e. the actions of different market participants, leads to certain market *outcomes*, which can be assessed using different types of efficiency measures in order to determine the overall *system performance*. Figure 2.9 visualizes these dependencies.

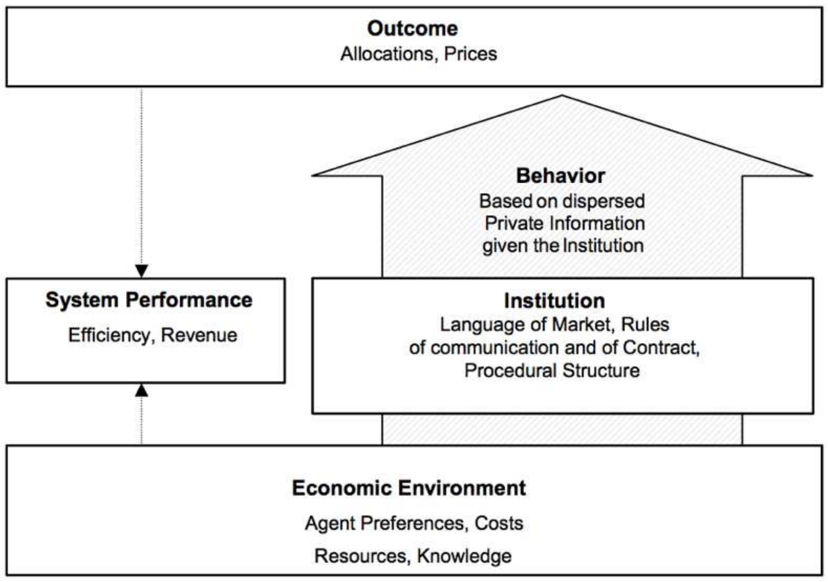


Fig. 2.9. Micro Economic System Framework (Source: Neumann, 2007, p. 9)

The objective of market engineering is to consciously design and adjust the institutions such that they influence the behavior of market participants in a way that leads to the desired outcome in terms of the performance metrics defined for the market under observation. The problem is that the definition and implementation of market institutions is complex and difficult. Consequently a *market engineering process* was proposed (Neumann, 2007), that aims at systematically supporting the persons or teams in charge with designing and implementing the market institutions. The process developed by Neumann consists of four main stages depicted in Figure 2.10, which are closely modeled after the traditional engineering design process of Pahl and Beitz (1984) and also resemble the Waterfall model in software engineering.

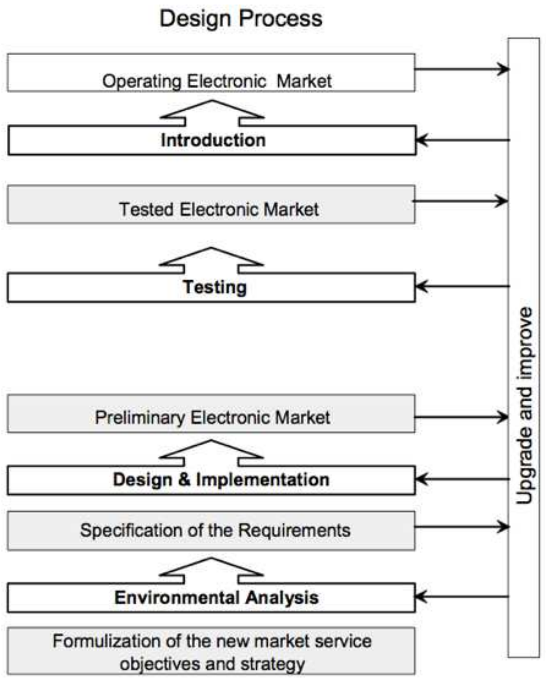

Design Process
Operating Electronic Market
Introduction
Tested Electronic Market
Testing
Preliminary Electronic Market
Design & Implementation
Specification of the Requirements
Environmental Analysis
Formulization of the new market service objectives and strategy
Upgrade and improve

Fig. 2.10. Market Engineering Design Process (Source: Neumann, 2007, p. 155)

1. *Environmental Analysis:* The environmental analysis is the first step of the market engineering process. In this phase the objectives and strategies of the future market owner need to be identified and defined first. These serve as guideline and framing for the market to be developed. Furthermore, strong emphasis is put to the analysis of the socio-economic market environment. Here the potential number and type of market participants, their preference structures, risk attitudes as well as the characteristics of the desired tradable goods and services are analyzed. With these information at hand functionalities and properties of the market (i.e. functional and non-functional requirements) can be determined.
2. *Design and Implementation:* Once the design problem is sufficiently described, the second phase of the market engineering process begins. Here, in three subsequent stages (i) a conceptual design, (ii) an embodiment design, and finally

(iii) a detail design are created. Based on these design documents the implementation of the market then takes place. For Neumann (2007) the *concept design* abstracts from details and mainly focuses on the design of the economic market rules and in particular on allocation and pricing rules as well as on the permitted bidding language and information disclosure rules in the market. During the *embodiment design* the abstract economic rules are then transformed into (semi-) formal process descriptions. The *detail design* then focuses on specifiying technical requirements (e.g. by detailing the required data formats and technical communication protocols). With these information at hand the system implementation is supposed to take place. For the structuring of the implementation phase itself Neumann (2007) refers to existing software engineering processes, in particular to V-model (Dröschel and Wiemers, 1999) described in more detail in Section 2.1.1.

3. Testing: Before the market platform is deployed and opened to the public the testing stage of the market engineering process has to be accomplished. In this stage Neumann (2007) describes different types of tests that need to be accomplished and that are mainly targeted at evaluating the economic performance of the market. Concrete testing methods recommended for this process step include (i) an axiomatic (analytical) evaluation of the market rules, (ii) and experimental evaluation of the participant' market bidding strategies and the resulting equilibria in laboratory or field experiments, and (iii) (agent-based) computational testing of the market.

4. Introduction: After testing is completed the market platform can be introduced to the general public. Necessary tasks in this stage are market monitoring, performance evaluation, and – where necessary – refactoring of market rules or system features in order to ensure market performance even under changing environmental conditions

A more detailed description of the market engineering process can be found in (Neumann, 2007, pp. 123-246).

In summary the *Market Engineering* process model provides detailed instructions and guidelines for the economic modeling and evaluation of markets on an economic level. It prescribes a process model that maps closely to the ideas of the Waterfall and BDUF models from software engineering. Also, while being very precise and detailed on the economic part of market modeling it falls short of providing detailed guidance on how to actually convert the detailed economic models into actually running market instances. Computer aided market engineering tools as described in the following subsections were designed to make up for these shortcomings by providing technical support to market engineers in designing, implementing and rolling out electronic markets.

2.3 Computer Aided Market Engineering Tools

While the *Market Engineering* process model provides value-add as a general guideline for creating electronic markets, complementing software tools were developed that were designed to support market engineers in designing, realizing, testing, and introducing electronic market places. Several such specific market engineering software platforms are known in literature. They all promise to provide a generic technical market infrastructure that is easily customizable by means of simple configuration rather than by programming, so that these platforms can be used to quickly realize wide ranges of different types of electronic markets. Out of the many different frameworks that were proposed (e.g. Chavez and Maes, 1996; Tsvetovatyy et al., 1997; Bichler et al., 1998; Dumas et al., 2004; Bartolini et al., 2005; Kersten and Lai, 2006) three are described in more detail in the following sections: *AuctionBot*, *GEM*, and *meet2trade*.

2.3.1 AuctionBot

AuctionBot was the first academic approach to build a generic market architecture. It was open for general public via Internet (Wurman et al., 1998) since January 1997. The main aim of this project was to develop a flexible, scalable, and robust auction platform. Figure 2.11 shows an overview of the AuctionBot architecture.

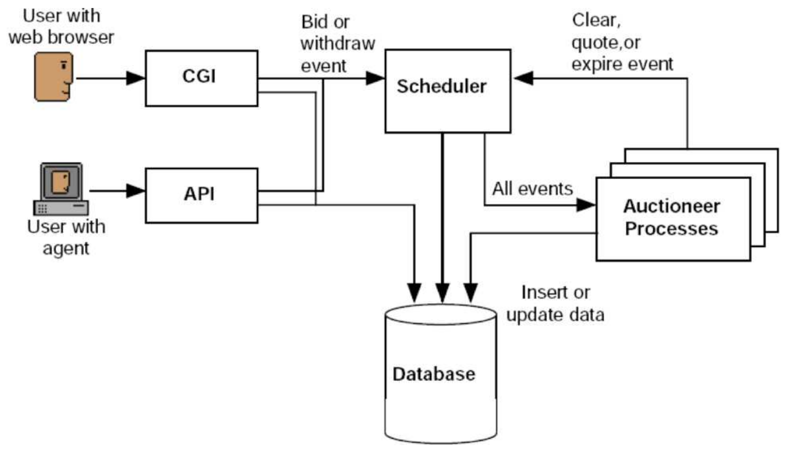


Fig. 2.11. Schema of AuctionBot's System Architecture (Source: Wurman et al., 1999)

AuctionBot supported several different types auction mechanisms and was designed to reduce the effort required for building and evaluating electronic market mechanisms before deploying them for commercial use. The support for different auction types was realized through a set of independent (orthogonal) design parameters that were identified as being common to most of the well-known auction types (Wurman et al., 1998). These parameters include time or event based start and stop rules, different types of limit price restrictions, visibility constraints for the orderbook, different pricing rules, different allocation rules etc. A sample AuctionBot configuration specifying a particular subset of parameters can be found in Appendix B. The theoretical concept of orthogonal market design parameters in AuctionBot was technically realized through a domain specific configuration language based on XML (Taylor and Wurman, 2000). AuctionBot has been inactive for several years. As of today neither its service nor its source code is available to the public anymore.

2.3.2 GEM

The "Global Electronic Market System" GEM is a follow-up approach to AuctionBot in building a generic market framework. *"GEM provides a flexible architecture that delineates the abstract components of a generic market and specifies control and data flow constraints between them, but allows variations in the concrete implementation of components with minimal or no impact on the implementation of other components."* (Rachlevsky-Reich et al., 1999). Furthermore the developers of GEM aimed at providing an infrastructure that allowed for pluggable market policy components which could be adapted and replaced during runtime, i.e. they designed the system such that market rules could be changed instantaneously during market operation in order to cope with changes in trading or system behavior.

On top of a Java based low level systems and communication infrastructure, GEM was build around three core components, a *"'Gate Keeper"*, a *"Trading Floor"*, and an *"Information Board"*. The *gate keeper* is the single point for order entry. It performs basic authentication and validation on the incoming orders and either rejects the orders or forwards them to the *trading floor* component as shown in Figure 2.12.

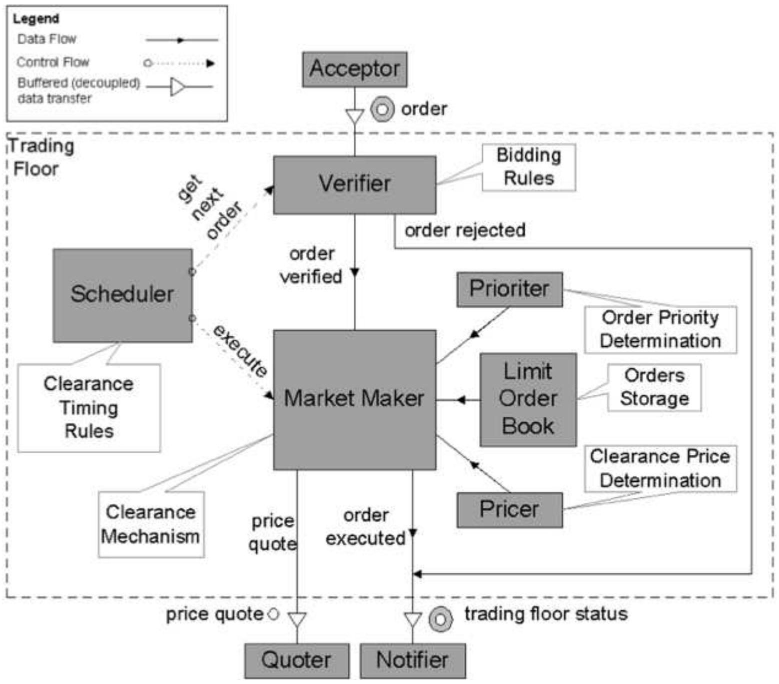


Fig. 2.12. Detailed view on GEM's "trading floor" component (Source: Rachlevsky-Reich et al., 1999)

The trading floor component is the heart of GEM's business logic and implements the market rules. Within this component the *verifier* first enforces specific market rules on incoming (validated) orders, e.g. like ensuring minimum bid increments between subsequent orders. Depending on the verification result orders are then rejected or passed on to the *scheduler*, which – depending on its configuration – forwards orders continuously or according to pre-specified timing rules to the *market maker*. The market maker is responsible for matching, pricing, and settling orders as well as for pushing (partially) unmatched orders back into the order queue for later execution. Activities of the market maker are monitored by the *information board*, which – depending on its configuration – may or may not disclose these information (i.e. quotes and trades) publicly in order to inform the market participants of current market (matching) events.

From a software engineering point of view the GEM developers achieved the desired modularity by applying the so called "bridge" pattern (Gamma et al., 1995) to the market specific business logic, which allows for a decoupling of the component's abstractions (technically represented through interfaces) from their concrete implementations. Like this, the implementation of a component could be changed or replaced without affecting implementation and internal states of the other components as long as the interfaces can be left untouched.

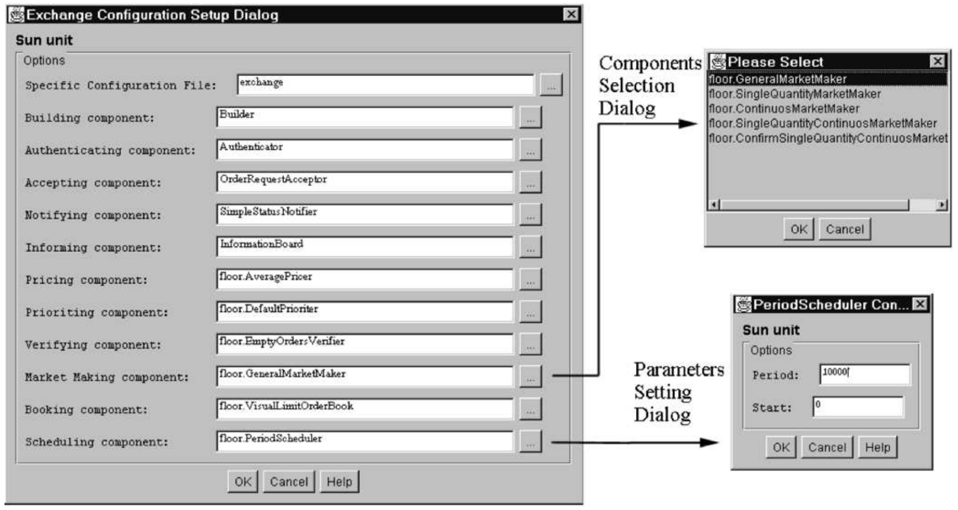
Rachlevsky-Reich et al. (1999) explicitly mention that they attempted to decompose the overall system into *"orthogonal components, whereby each component encapsulates a distinct aspect of the market policy that can be customized independently of other components, and is governed by a separate set of rules"*. Figure 2.13 shows the configuration manager of GEM, which is used to tie together the different distinct components into a running overall system. In the configuration manager the system administrator can choose the concrete instance of a particular component to be used during market execution, e.g. a `PeriodScheduler` instance as scheduler. Each of the chosen components can then be further parameterized. In the depicted case the scheduling interval for successive market matchings can be adjusted, e.g. to 10,000 milliseconds or 10 seconds between to matching operations.

The GEM project was completed nearly ten years ago and did not experience any further development since then. Its design, architecture, and findings are available in literature but neither the source code nor the project website are available anymore and also an interview request remained unanswered.

2.3.3 meet2trade

meet2trade was designed and developed as a part of the electronic financial brokerage project that was carried out at the Institute of Information Systems and Management at the University of Karlsruhe in the time between 2000 and 2004, together with several industry partners such as the Stuttgart Stock Exchange, Reuters AG, and trading fair AG. One of the main objectives of the project was to develop a new and progressive tools for electronic trading (Weinhardt et al., 2006).

meet2trade is a software suite that consists of several applications as shown in Figure 2.14. The core functionality is provided by the dynamically configurable auction



Exchange Configuration Setup Dialog

Sun unit

Options

```
Specific Configuration File:   exchange
Building component:            Builder
Authenticating component:      Authenticator
Accepting component:           OrderRequestAcceptor
Notifying component:           SimpleStatusNotifier
Informing component:           InformationBoard
Pricing component:             floor.AveragePricer
Prioriting component:          floor.DefaultPrioriter
Verifying component:           floor.EmptyOrdersVerifier
Market Making component:       floor.GeneralMarketMaker
Booking component:             floor.VisualLimitOrderBook
Scheduling component:          floor.PeriodScheduler
```

Please Select

```
floor.GeneralMarketMaker
floor.SingleQuantityMarketMaker
floor.ContinuosMarketMaker
floor.SingleQuantityContinuosMarketMaker
floor.ConfirmSingleQuantityContinuosMarket
```

PeriodScheduler Con...

Sun unit

Options

```
Period:  10000
Start:   0
```

Fig. 2.13. GEM's configuration interface (Source: Rachlevsky-Reich et al., 1999)

runtime environment (ARTE) that supports a large number of auction types to conduct trading. In the following listing the functionality of the different components is summarized.

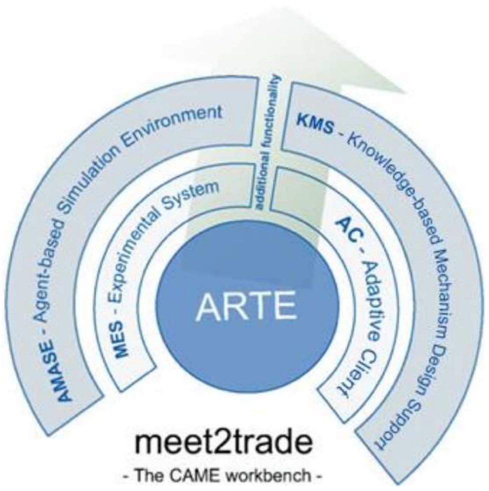

ARTE
MES - Experimental System
AMASE - Agent-based Simulation Environment
additional functionality
KMS - Knowledge-based Mechanism Design Support
AC - Adaptive Client
meet2trade
- The CAME workbench -

Fig. 2.14. The meet2trade market engineering tool suite (Source: Block and Neumann, 2008)

ARTE – Auction run-time environment: The design of market mechanisms in meet2trade is based on a parameterization approach - i.e. a large variety of exchange mechanism (mostly auctions) can be described by a set of parameters representing their specific rules. ARTE is responsible for creating market mechanisms defined by XML instances that contain the required sets of parameters.

Hence ARTE is at the core of the computer-aided market engineering workbench. A configuration editor facilitates the generation of XML instances and also provides a convenient mechanism to upload them into ARTE (Mäkiö and Weber, 2004).

AC – Adaptive Client: The main user interface of the CAME tool suite is the adaptive client (AC). Its graphical user interface is remotely configured by the ARTE core in order to present a suitable user interface for a specific mechanism as defined in the respective XML instance. The adaptability of the client is a key enabler that allows the dynamic rendering of different GUIs according to the needs of specific mechanisms.

AMASE – Agent-based market simulation environment: AMASE is an agent- based simulation environment, which allows the automated testing of market mechanisms. Simple test scenarios can be produced on-the-fly, while more complex scenarios require some coding of the agent behavior (Czernohous, 2005; van Dinther, 2006). AMASE renders predictions on how market mechanisms will perform using simulation techniques that allow valid predictions even about sophisticated market mechanisms.

MES – Market experiment shell: In order to examine specific procurement mechanisms, an experimental system has been added to the meet2trade software suite. The main objective is to conduct experiments on the original system instead of replicating and running the mechanism in experimental software. This approach facilitates experimental studies since the market has to be modeled only once within meet2trade and avoids potential biases from the usage of different user interfaces. For experiments the standard AC client is running in experimental mode, which enables more detailed logging of user actions and allows tight control over permitted actions in different stages of the experiments (Kolitz et al., 2007).

KMS – Knowledge-based Mechanism Design Support The KMS knowledge based was contributed to the meet2trade tool suite only after the main project was already finished. It was developed as a proof-of-concept prototype by the author of this thesis as one of the deliverables of the ANEGOM project and is described in more detail in Section 4.2.1.[6] The initial motivation for the development of KMS was the insight that an increasing degree of freedom and options in the configuration of auctions through MML makes it difficult to choose an appropriate set of parameters for a given environmental situation. KMS is a knowledge base that uses case base reasoning to compare a (parameterized) description of the respective market environment with locally stored expert recommendations, where each of the stored recommendations is predicated itself with a set of parameters. A user of KMS can request recommendations on which mechanism to choose best by submitting a parametric search query. Stored recommendations with the best matching parameter overlap are returned sorted by matching quality (i.e. relevance) thus providing the user with concrete reference on which market configuration with which parameter values should be applied (Neumann et al., 2007).

[6] `http://www.im.uni-karlsruhe.de/anegom`

Technically meet2trade is designed as a three-tier architecture, which is depicted in Figure 2.15. A client layer with different Java Swing trading clients resides on top of the software stack. Below, in layer two, the business layer with the ARTE core is positioned, which in turn builds on top of a database layer that provides an abstraction to the underlying persistence technology and also provides common base functionality like logging services.

The ARTE core is implemented based on Enterprise Java Beans (Monson-Haefel and Weissinger, 2003) and communicates with the trading clients via asynchronous Java Message Service (JMS) messages based on XML. The parametrization of the markets is modeled through a domain specific modeling language called Market Modeling Language (MML), which is based on XML and contains about 110 parameters allowing for a detailed market design and configuration (Mäkiö and Weber, 2005). A sample MML configuration can be found in Appendix B.

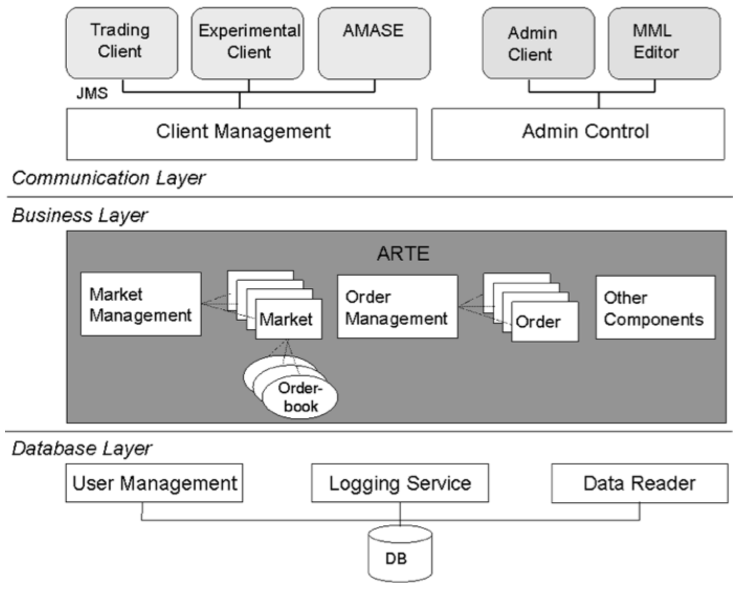


Fig. 2.15. Technical architecture of meet2trade (Source: Weinhardt et al., 2006, p. 29)

A distinct feature of the meet2trade tool suite is its comprehensive functionality and flexibility. meet2trade supports a (comparably) much wider range of basic auction types then GEM or AuctionBot. In parts, this flexibility stems from the implemented concept of "meta-markets", i.e. hybrid market instances that are constructed out of a set of simple market instances. This abstraction allows, for example, the modeling of a typical procurement auction process that consists of an initial sealed bid first price auction (RFQ) and a subsequent english auction in which only a winning subset of bidders from the initial RFQ is allowed to participate in.

The meet2trade tool suite is no longer under active development. Instead, around 2006, the team at IISM started investigations for a successor platform that takes the lessons learned from building meet2trade into account in order to eventually take over as the standard platform for market engineering research and development at the institute. In the next chapter, the insights from building and using meet2trade in combination with the market engineering process models described before and some of the difficulties the team experienced with the generic platform approach experienced are summarized. Based on these lessons learned and based on a use case analysis conducted in Section 3.3.2 a set of requirements for such a “next generation market engineering platform” are formulated.

2.4 Summary

In summary, significant research work has been devoted to developing methodologies and supporting software artifacts aimed at supporting the development of electronic markets from generating new market ideas to assessing existing market environments, to creating an evaluating economic market models, to realizing and running markets in practice.

Still, none of the analyzed market engineering platforms is actively supported and developed anymore. Also the existing market engineering process models seem to be closely modeled after traditional Waterfall-style software engineering models. These in turn proved to be problematic in software engineering practice and are now largely displaced by newer, agile software engineering process models.

Taken together, all these observations call for a systematic review of market engineering process models in the light of agile methodologies and for the (re-)development of new supporting market engineering software artifacts that go along well with agile development practices and principles. Consequently, the next chapter is devoted to formulating requirements that an agile market engineering process model and supporting software artifacts need to fulfill to finally bridge the gap between abstract and theoretic market mechanism models on the one hand and publicly running electronic markets on the other.

3

Requirement Analysis

In this chapter, requirements for an agile market engineering process model and for supporting artifacts are determined. First, problems and shortcomings of existing models and tools – as described in the previous chapter – are summarized. Subsequently a use case analysis is conducted that aims at detecting key stakeholders and key requirements for the process model to be developed as well as for its supporting artifacts. Conclusions from both sections together are then used to formulate a set of requirements that the agile market engineering process needs to address on a technical as well as on a managerial (i.e. business) level.

3.1 Assessment of existing market development process models

With the Process / Stakeholder Benefit Framework, Kambil and van Heck (2002) developed a methodology to systematically assess a market environment, to develop new market ideas, to asses risks and potentials of the idea, and to then develop a strategic plan for entering the market. Additionally, the authors provide many real life examples on how markets have been implemented or changed in the past in different industries and under different environmental conditions. All this provides valuable support for the initial stages in the development of new markets (ideas). However, apart from case study descriptions, the authors do not describe the next logical steps, i.e. a process on how to proceed after a new market idea was developed and its market potential was estimated and assessed.

Here, the Market Engineering model of Neumann (2007) provides useful guidance. It particularly describes a process for developing an economic mechanism design (i.e. the market rules) of a new market taking into account market environment, tradable product(s), and expected types of market participants and their behavior. Still, this process model falls short of appropriately describing the logical next process step on the way to creating a running electronic market: Concrete and detailed guidance on how the previously developed economic market model should be translated into an actually *running* market instance would be required. Neumann (2007) only points to *"state-of-the art approaches like the V-model [...] supplemented by methods such* *as the FUSION [...] and tools such as UML[...] such that the software engineering process will not further be elaborated."* This referral is insufficient and additionally points to a software engineering process model that is outdated in its core ideas and is superseded by more performant and agile process models (see Section 2.1).

In short, the two models of Kambil / van Heck and Neumann / Weinhardt provide support for developing and strategically positioning new market ideas, and for modeling them on an economic level. Almost no guidance is provided on how running markets should be developed based on these inputs. This conceptual gap is addressed in this thesis.

3.2 Assessment of existing market engineering software artifacts

Neither for AuctionBot nor for GEM source or even binary code was available for analysis and consequently the code work of these projects was not assessed. Still, code from the meet2trade project was available as source and binary code and is analyzed in more detail in the following paragraphs.

One of the lessons learned from meet2trade is that the extensive genericity of the market framework (especially in the ARTE core and in the adaptive clients) is a useful abstraction. But this holds only as long as it could be used "as is" and proved to be a significant burden as soon as functional extensions beyond the configuration space are required. For the two main real world projects (NorA, see Chen et al. (2006) and Stoccer, see Luckner et al. (2005)), where meet2trade was used as market platform, the configuration space was considered "sufficient" during a first requirements appraisal but still proved to be too small later on, when detailed requirements of the markets were defined.

STOCCER is a prediction market system for soccer events where soccer fans can trade virtual stocks of national and international teams and in this manner predict the outcome of a tournament as precisely as possible. Fore the implementation, a continuous double auction model was chosen as the underlying economic market mechanism. Setting up a corresponding market instance was easy to accomplish in meet2trade by simply creating an appropriate MML configuration document and by then deploying it to the ARTE runtime environment. But, additionally to this basic market model, a scoring function for market participants based on a combination of weighted portfolio values and cash positions was required as winner determination mechanism for the prediction market. Furthermore, the user interface had to be web-based in order to provide easy access to users on the Internet. Both requirements were "beyond" the MML configuration space and, thus, had to be added manually. The corresponding extensions and refactorings required extra efforts of about 1.5 years FTE.[1]

In the case of the NorA project, a lab-experiment based study on the influence of mechanism and system design on market outcome, a multi-attribute english reverse

[1] FTE is the abbreviation for Full Time Equivalent. A task which requires an effort of 1 month FTE means that 1 person works for one month full time to accomplish the task.

auction was required as market mechanism. While a regular English reverse auction was covered by MML design space, the multi-attribute version of it was not. As in Stoccer before, the required refactorings in this project proved to be difficult and time consuming with an estimated effort of about 8 FTE months.[2]

In both projects, the required adaptations and extensions turned out to be difficult to manage. This is mainly because the adaptation of the ARTE runtime environment with its high inherent complexity of the core architectural elements was difficult to extend and required a very detailed knowledge of the underlying software architecture.

To quantify the complexity of the source code of meet2trade, and in particular of the ARTE runtime environment, the classes from the `org.efits.Trade` package that implement most of the market logic were analyzed.[3] Based on the source code of these eight classes, common software metrics such as cyclomatic complexity (CC) of the functions of each class, number of non-commentary lines of code (LOC) and number of methods and attributes per class were derived. Several empirical studies indicate that these metrics provide a good prediction for the effort of software maintenance. McCabe (1976) claims, that based on his empirical findings, CC of modules and functions should not exceed an index value of 10 (though he admits that this value was chosen somewhat arbitrarily). Based on further studies Kafura and Reddy (1987) and Grady (1994) come to the conclusion that CC should not exceed 14 in order to keep software well maintainable. Prechelt et al. (1999) show that the number of methods in a class have a high correlation with the maintenance effort. Shepperd (1988) claim a similar correlation between maintenance effort and LOC.

Table 3.1 lists the above metrics for each of the eight core classes of meet2trade, sorted by decreasing cyclomatic complexity (column 2). The analysis was performed using Cyvis[4].

This overview gives a sense of the inherent complexity that the core elements of meet2trade possess. The cyclomatic complexity of the classes `org.efits.Trade.Order.Order` and `org.efits.Trade.Markt.AbstractMarkt` is within the recommendations. Still these classes exhibit a very large number of methods and attributes. The most complex class, `org.efits.Trade.Markt.Allocator_CDA_III`, has a cyclomatic complexity that is 13 times higher than the recommendation of Kafura and Reddy (1987) and Grady (1994). This can be seen as a root cause for the difficulties that the team members of the Stoccer and NorA project experienced while trying to extend meet2trade. The code complexity of meet2trade can be seen as the price for its genericity.

[2] If a web-based interface would have been required for the meet2trade instance in NorA as it was the case in Stoccer efforts would have been significantly higher.

[3] In fact the complexity analysis was conducted for the complete ARTE core of meet2trade, which consists of an overall of 13,811 methods, written in 847 classes, and divided into 92 packages. The raw report consists of 349 pages so that only the results for the core market logic is presented within this work. Full results of the analysis can be obtained from the author on demand.

[4] http://cyvis.sourceforge.net

Table 3.1. Software quality metrics for the meet2trade core classes

Class	max. CC	LOC	#Methods
`org.efits.Trade.Markt.Allocator_CDA_III`	185	842	4
`org.efits.Trade.Orderbuch.Orderbuch`	103	2998	156
`org.efits.Trade.MetaMarkt.MetaMarkt`	61	3368	282
`org.efits.Trade.Markt.Allocator_CallMarket`	54	276	2
`org.efits.Trade.Order.OrderManager`	33	789	42
`org.efits.Trade.Markt.Market`	30	876	20
`org.efits.Trade.Order.Order`	8	913	123
`org.efits.Trade.Markt.AbstractMarkt`	7	837	192

Also, meet2trade seems to have been affected by another problem. Selby (2007, p. 168) states that *"words are cheap. During the specification phase, it is all to easy to add gold-plating functions to the product specification, without a good understanding of their effect on the product's conceptual integrity or the project's required effort."* With generality as one of the key value propositions, the requirements specification had to cover a great variety of market models in a broad range of potential application scenarios (at least initially) without any concrete use cases or any concrete projects to develop against. Like this, more and more gold-plating features were added over time for "potential future use" that lead to more and more code complexity on the one hand and decreased maintainability and extensibility on the other. Still, for all of the concrete application scenarios, where meet2trade was used (see also Section 2.3.3), the design space of the market modeling language proved to be not "large" enough in one or more particular aspects and the resulting code adaptations and extensions were found to be time-consuming and complex.

This is why Selby (2007) advocates the development of "simpler software". In an experimental study comprising seven projects he compared software engineering projects that used an extensive upfront specification process with projects that used rapid prototyping and software delivery in small incremental versions with frequent user feedback.

He reports that all projects delivered software artifacts that were roughly equivalent in performance though agile projects reportedly required about 40% fewer LOC and approximately 40% less development time. His conclusion is that prototype driven projects initially provide only the very core functionality and are only extended where "necessary" after inspecting the running systems. On the contrary, in specification driven projects, some of the features added to the requirements specification are not absolutely necessary.

Two key insights can be extracted from the previous analysis:

- The two existing market engineering process models provide good guidance for developing markets up to an economic (mechanism design) model but lack support for actually getting markets *running*

- Generic market platforms tend to be complex by design and are thus difficult to maintain and to extend. Simpler approaches on the other hand might be able to deliver the same functionality at lower cost and complexity.

With the insights at hand the following sections focus on developing use cases and identifying important stake holders for a simple and efficient market engineering process model that bridges the gap between theoretic models and running market.

3.3 Use Case Analysis

This section is focused on identifying involved persons (stake holders) and typical tasks (use cases) that occur during the development of a new market.

3.3.1 Stakeholders

Design and implementation of (electronic) markets require the involvement of multiple parties in several different roles and with several different professional backgrounds. In this context a “role” is considered as a set of responsibilities and privileges that one or more persons in this role take over. The following analysis differentiates five core roles. Each of them is described separately but it should be noted that a single person can embody several roles. E.g. a person can be *business owner* and *market expert*. Vice versa, a single role can be assigned to several persons as well. E.g. the role of the *market developer* is likely to be assigned to a team of several persons. The following listing provides a short description for each of the roles.

1. The *Business Owner (BO)* is the role of the person(s) who develop the initial business idea. The BO is the driver for the realization of the market on the management side. In reality, the business owner can be the customer who pays for the development of the market itself. Alternatively, a proxy (e.g. a key account manager of the market development firm) can take over the role so that the customer does not have to be personally present all the time.
2. The *Market Expert (ME)* role is that of a specialist. Mechanism Design is a specific research area within economics that *“takes a systematic look at the design of institutions [i.e. market rules] and how these affect the outcomes of interactions [between market participants].”* Jackson (2000). The ME is trained in economics and particularly in mechanism design. The ME supports the business owner and the market developer(s) in defining market rules, i.e. allocation rules, pricing rules, market language, and information dissemination policies according to sound economic principles, scientifically founded research findings and / or results from (game) theoretic analysis.
3. The *Market Developer (MD)* is concerned with the technical realization of the market system, i.e. with implementing the business logic and the market rules into an executable electronic market platform.
4. The *Market Operator (MO)* technically operates the market by deploying the market platform on a suitable server, connecting it to other required services

like e.g. a database or an email server. Also the MO continuously observes and maintains the market on a system level e.g. by monitoring log files, processor and memory consumption, and by escalating runtime errors to MDs.

5. The *Market Participants (MP)* are end customers who conduct transactions on the electronic market platform buying or selling products or services, and who consume the information it provides.

Additionally, the *Change Manager (CM)* is a very important role though the CM is not assuming any functional tasks directly related to the market development. Instead this role represents a change agent (Hutton, 1994) and is well comparable to the Scrum Master Role (Schwaber and Beedle, 2001) in the Scrum software engineering process model (see Section 2.1.2). The CM is specifically focused on detecting and resolving impediments within the project organization, within the development process, in the interaction between team members, and in the interaction with the project's environment (e.g. with customers or other departments within the organization). The CM does not occur in the use cases described in the following section.

3.3.2 Use Cases

Based on the previous analysis and based on practical insights from several market engineering projects the following set of use cases is developed. The use cases are divided into three different phases *Pre-Development Phase*, *Market Development Phase*, and *Market Operation Phase*. The pre-development phase roughly comprises market assessment and theoretic market modeling activities as covered by the frameworks of Kambil and van Heck (2002) and Neumann (2007). The development phase comprises all tasks related to the technical creation of the market platform, while all tasks pertaining to running the market in reality are described in the operation phase.

Pre-Development Phase

During the pre-development phase the idea for a new (electronic) market is developed and high-level requirements are deduced. The results from this phase serve as input for the market development phase.

Use Case 1 (Develop Market Vision and Assess Market Environment)
The process starts with a detailed analysis of the particular market environment, its stake holders and interest groups, potential areas for process improvements, or other forms of value creation. With these analysis at hand, the business owner is able to describe the vision of the market and its business model.

Use Case 2 (Develop Abstract Mechanism Design)
With the vision and the business model in mind the BO – supported by the ME – can start developing a theoretical market model taking into account insights from economics and in particular from mechanism design, e.g. to determine likely market equilibria or expected dominant behavioral patterns (strategies) of market participants and from mechanism design. The result will be a set of (abstract) market

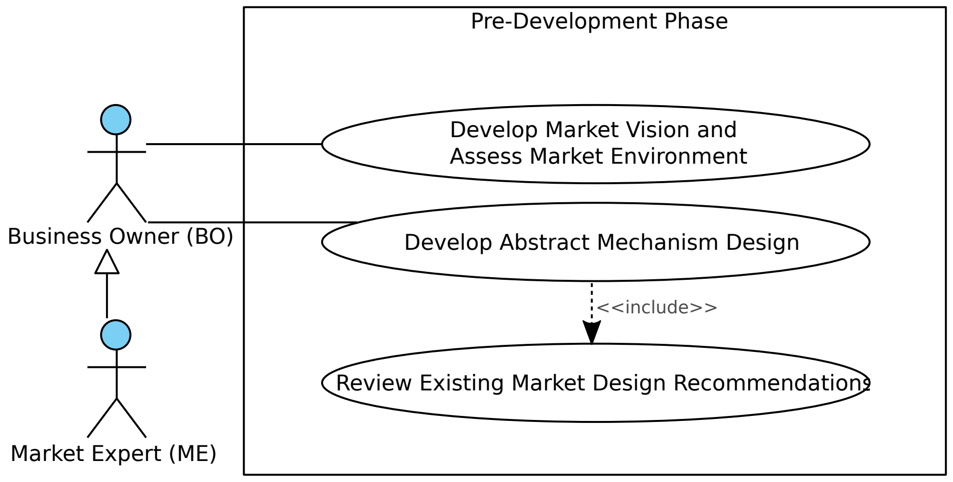

Pre-Development Phase
Develop Market Vision and Assess Market Environment
Develop Abstract Mechanism Design
<<include>>
Review Existing Market Design Recommendations
Business Owner (BO)
Market Expert (ME)

Fig. 3.1. Use Cases in the Pre-Development Phase

institutions, i.e. allocation rules, pricing rules, bidding language, and information dissemination policies.

Use Case 3 (Review Existing Market Design Recommendations)
Part of the previous use case is the review of the chosen mechanism design. Here, BO and ME need to relate their (preliminary) market design to other already existing designs and findings from research in order to gain insights on how market participants will act once the market is publicly accessible.

Development Phase

Based on a clear understanding of the market vision and the business model, and with an abstract economic mechanism design concept including expertise from related markets and corresponding research findings at hand, the market development phase starts. In this phase, the technical market system will be developed in short incremental cycles. These cycles allow for a flexible realignment of requirements in reaction to insights on market performance based on hands-on experiences and empirically observed market dynamics from the running market system.

Use Case 4 (Describe / Update / Detail Requirements)
At the begining of the market development phase BO, MO, and ME need to develop a joint understanding of the market vision (i.e. the overall goal to achieve). With this common understanding an initial set of requirements needs to be developed.

Use Case 5 (Search and instantiate existing market as template)
In the first iteration of the the market development phase, the MD reviews already existing market instances. His aim is to detect previously developed markets that

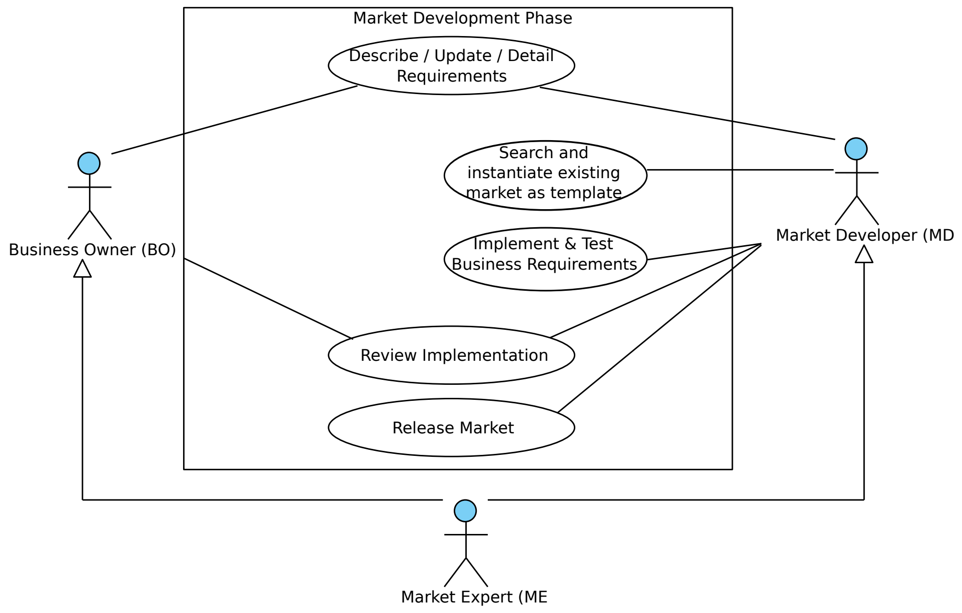

Market Development Phase
Describe / Update / Detail Requirements
Search and instantiate existing market as template
Implement & Test Business Requirements
Review Implementation
Release Market
Business Owner (BO)
Market Developer (MD
Market Expert (ME

Fig. 3.2. Use Cases in the Market Development Phase

are deemed "sufficiently" similar in their market model[5] so that they can serve as the technical basis for the development of the new market.

Use Case 6 (Implement and Test Business Requirements)
Usually a "good fitting" existing market should already implement a significant subset of the required technical and business functionality of the new market. Missing features have to be implemented by extending or replacing the existing code artifacts where necessary.

Use Case 7 (Review Implementation)
Once the set of business requirements is implemented, tested and running live on a testing system BO, MD, and ME jointly conduct a review based on the running system. The experiences and insights from this live review then become the basis for the formulation of new requirements / features, or for requirement changes. The review ends with an approval (or, if critical issues are detected with a reject) of the new, implemented feature set.

Use Case 8 (Release Market)
After the review process is finished and the implemented feature set was approved, a new version of the market is released. A new market release essentially triggers a redeployment of the revised market on the production systems as described in the following use cases.

[5] For example in Chapter 5 the development of several prediction markets is described that all use a double auction mechanism and are base on a simple double auction market platform developed before.

Operations Phase

After an instance of the electronic market is released at the end of a development phase a new market operations phase begins.

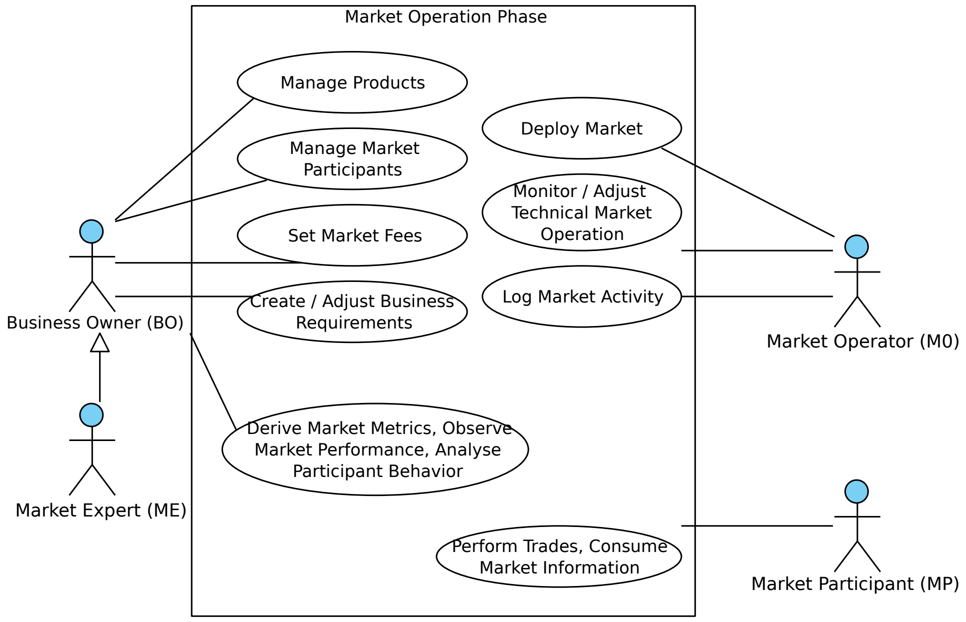

Market Operation Phase
Manage Products
Manage Market Participants
Set Market Fees
Create / Adjust Business Requirements
Derive Market Metrics, Observe Market Performance, Analyse Participant Behavior
Deploy Market
Monitor / Adjust Technical Market Operation
Log Market Activity
Perform Trades, Consume Market Information
Business Owner (BO)
Market Expert (ME)
Market Operator (M0)
Market Participant (MP)

Fig. 3.3. Use Cases Market Operation Phase

Use Case 9 (Deploy market)
The market operator installs the executable code of the electronic market on the production server infrastructure. Sometimes, especially in early development, deployments can be made to a "staging" environment, a second production environment that contains sample data and allows for intensive testing under realistic conditions without influencing operations of the production environment. For both deployment types, the MO ensures that all necessary runtime components are configured correctly including e.g. configuration for database access and email connectivity and that the market is accessible online.

Use Case 10 (Manage products)
The business owner creates, edits, removes, and enables or disables products and the corresponding trade activities.

Use Case 11 (Manage market participants)
The business owner grants or restricts access to products, and controls orders, depot positions and account settings. In other words, the business owner exerts market control in order to ensure smooth market operations and compliance with market rules.

Use Case 12 (Set market fees)
The business owner can introduce and adjust market fees for (i) general market participation or (ii) trading activities.

Use Case 13 (Derive Market Metrics, observe performance)
The BO – supported by the ME – observes the economic performance of the market by analyzing occurred trades and by deriving appropriate market metrics like implicit transaction costs, product prices, transaction volume or liquidity. These insights can then serve as input in form of new requirements and change requests for subsequent market development phases.

Use Case 14 (Create / Adjust Requirements)
Based on observations of market activity and MP behavior, the BO – supported by ME and his market performance analysis – formulates new or adapts existing requirements.

Use Case 15 (Monitor / adjust technical market operation)
The MO continuously observes and maintains the market by monitoring log files and processor and memory consumption. He reports runtime errors to MD where required.

Use Case 16 (Perform trades, consume market information)
Market participants get information about tradable products and product categories, they place orders, view their trading history, depot position and money account transactions, and edit their personal user details.

3.4 Requirements for an Agile Market Development

Several articles in literature postulate requirements that electronic markets need to address. In the following paragraphs, a short overview of these requirements is compiled, which provides insights on what type of functionality is commonly considered important for electronic markets.

According to Malone et al. (1987) electronic markets need to provide easy interconnectivity with other applications or services and multi-channel access. They should also facilitate automated information aggregation providing some sort of automated decision support to customers during product and offer assessment and selection. Their reasoning builds on theory of Coase (1937) who claims that a market will be the coordination mechanism of choice if its transaction cost (e.g. cost for searching or matchmaking) are lower than those of a firm (i.e. hierarchy), which is set up to serve the same purpose. Consequently, Malone et al. see a market's ability to reducing transaction cost as core requirements for its success.

Schmid (1997) follows this argumentation and notes that most importantly transaction costs for product browsing and searching need to be kept low. In other words, providing transparency is a key requirement for electronic markets. Additionally, Schmid argues that functional components for conducting negotiations, for supporting contracting, and for accomplishing settlements are key functionalities of a market and always required. Furthermore, negotiation rules[6] need to be easy to define and to adjust. Last no least, he notes that support for calculating logistics cost, possibilities to store meta-information on products (e.g. consumer feedback or ratings), and decision support for customers selecting and evaluating offers on the market would be desirable.

Bichler et al. (1998) note that transparent listing of products, prices and historic transactions is a key requirement for electronic markets. They also argue that the availability of human or electronic brokers (mediators) can be an important value-add for markets as they can provide assistance during the product search / discovery phase as well as during subsequent negotiations. Furthermore, Bichler et al. state that brokers might take on the role of market facilitators and serve the e.g. as liquidity providers in financial markets.

In contrast to Bichler et al., Segev et al. (1999) claim from the observation of business-to-business procurement markets that complete market transparency would rather deter possible bidders (suppliers) from market entry as they fear stiff price competitions that do not leave them room for differentiating themselves from competitors via value adding services. Thus, they argue that limited market transparency is a key success factor in the procurement domain and can be seen as a price to be paid for attracting participants. On the other hand, the authors admit that the ability to transparently find and reliably authenticate potential contracting partners can be an important attractor as well. Lastly, Segev et al. point out that electronic markets on the business-to-business level need good interfaces for human users but should also provide technical interfaces that allow connections to and integration of other systems (e.g. ERP systems of market participants) and data sources (e.g. third party catalogue data).

In summary, an optimal degree of market transparency seems to be difficult to determine and dependent on the application domain. Still, the authors agree, that (i) good human user interfaces that facilitate and ease product search, (ii) technical market interfaces for machine-to-machine communication, (iii) decision support for product choice and during negotiations, and (iv) support for contracting and settlement are amongst the most important requirements that electronic markets need to fulfill.

Based these recommendations, based on the review of existing market engineering process models, and based on the uses case analysis from the preceding section the following requirements are deduced.

3.4.1 Business Requirements

Requirement 1 (Adjustable support for common market functionality) Common market functionality comprises the definition of (i) a market language, (ii) an allocation mechanism, (iii) a pricing mechanism, and (iv) rules that pertain

[6] In the nomenclature of Schmid negotiation rules include rules for market communication, permitted types of market participants as well as their respective rights and obligations on the market.

to information flow and information disclosure (market transparency) (see Anandalingam et al., 2005, p.322). An electronic market needs to implement all of these elements. But as the literature analysis in the preceding section shows the particular requirements might change over time and are likely to be dependent on the respective market domain. Thus extending and adjusting these core market components should be simple to accomplish.

Requirement 2 (Authentication & Authorization)
An electronic market needs to support secure and trustworthy user authentication (see Segev et al., 1999) that allows market participants to believe in the claimed identities of the contracting counterparties, which is necessary to create trust and thus to foster trade.

Additionally, electronic markets need to allow for at least two basic authorization roles or profiles: *market participant* and *business owner* (see use case analysis in Section 3.3.2). Market participants will have restricted access to the electronic market in that they are only permitted to (i) access the market's information facilities, (ii) learn about tradable products, (iii) view the market history, (iv) participate in trading activities, and (v) administrate their own personal data. In contrast, members of the business owner role are usually not authorized to trade in the market but instead are able to manage the trading platform, i.e. to activate or suspend trading of different products, to set and adjust market fees, to manage the catalogue of tradable products, to verify and adjust user accounts and inventories, and to monitor market activity.

Requirement 3 (Integrated & adjustable information services)
Beside the core functionality of a trading platform, the markets should provide basic information services, in order to offer relevant pre-trade and post- trade information for market participants, like the state of the order book of a product or current product prices effectively reducing transaction cost for information retrieval. The exact information type and granularity should be easily customizable (see discussion at the beginning of Section 3.4).

Requirement 4 (Support for multi-channel market access)
Market information services as well as core trading functionality should be available via web-based user interfaces. Optionally, it should be possible to expose these functionalities to different communication channels (e.g. web services, AMQP, or E-Mail). Web Service access will be particularly important for automatic trading agents and external decision support systems (see Malone et al., 1987; Segev et al., 1999).

Requirement 5 (Optional decision & automation support)
It should be possible to easily integrate decision support functionality dedicated to both product selection support, and / or negotiation support into the electronic market (see Schmid, 1997; Bichler et al., 1998).

3.4.2 Technical Requirements

Requirement 6 (Simple software architecture)
To enable rapid development and customization of market templates, a software architecture common to most types of electronic markets should be created. This architecture needs to be as simple as possible (see discussion in Section 3.2) but still flexible enough to easily allow adding new or removing unnecessary architectural elements (such as decision support, see discussion in Section 3.4). Also the modification of existing components in the architecture should be easily possible and the communication scheme between different components should be easily adjustably where necessary.

Requirement 7 (Reusability of markets)
A market instance built on top of the market framework described in this thesis should be fully functional, providing general functionality of a trading platform. Besides core components like a task scheduler a clearing module, or a settlement component, market instances should also contain an integrated user and product management (see use case analysis in Section 3.3.2). All these facets of the markets should build on (carefully chosen) defaults and conventions that allow for easy reuse in later projects.

3.5 Summary

In the first part of this chapter, existing market engineering process models as well as existing market engineering software artifacts are analyzed in order to identify gaps and shortcomings in these previous works. Subsequently, a use case analysis is conducted in order to identify important stakeholders as well as typical tasks (use cases) that frequently occur during the development of a new market. Based on these insights a set of requirements is formulated that an agile market engineering process model needs to address. With this set of requirements at hand the following chapter describes a market architecture as well as a market development process model that strives to fulfill these requirements.

4 Agile Market Engineering

Based on the requirement analysis in Chapter 3 an agile market engineering process model together with a set of supporting software artifacts is described in this chapter. The combination of both, process model and software artifacts, is designed to address the shortcomings of existing market engineering process models and software tools identified in Chapter 3. Building on previous work of Beck (1999); MacCormack (2001); Cohen et al. (2004); Cockburn (2004); Poppendieck and Poppendieck (2006) the proposed model is an adaptation and specification of general purpose agile software engineering practices for the domain of market engineering Cao et al. (2009). This process model is designed to particularly fit relatively small project teams (2 – 7 persons) who...

1. develop and release market platforms in short and time boxed cycles
2. have change tolerance in their minds and in their technical market architecture and thus frequently reflect and (whenever necessary) adapt the developed market as well as the market engineering process itself as new ideas and insights grow over time
3. work physically close to each other (in one large room or at least in adjacent rooms)
4. make intensive use of whiteboards, flip charts, informal communication processes, and feedback from end users to develop, discuss and document ideas and solutions
5. have market (mechanism design) experts within their team or at least in close vicinity so that frequent communication is ensured
6. can communicate openly within the team without fearing any type of repressions
7. employ test automation, continuous integration, and automated software dependency management in combination with rapid application development frameworks

The proposed agile market engineering process model *can* be understood and used as a strict process prescription for a market engineering project. In this case the respective project team can simply follow the process steps shown in Figure 4.1 from top to bottom and from left to right. However, this is not how this model is intended to be used. Rather than strictly prescribing or even enforcing particular

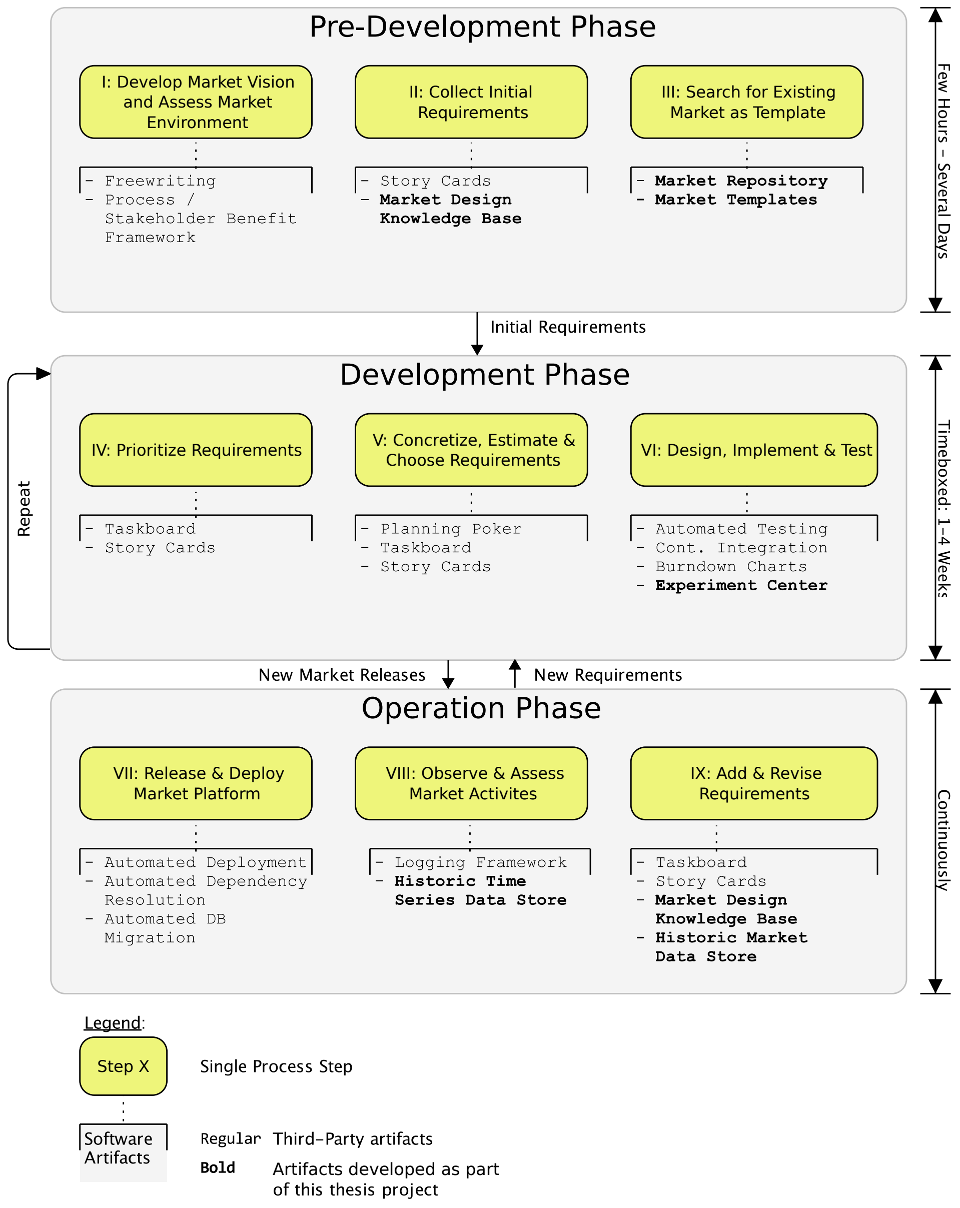

Pre-Development Phase
I: Develop Market Vision and Assess Market Environment
- Freewriting
- Process / Stakeholder Benefit Framework
II: Collect Initial Requirements
- Story Cards
- **Market Design Knowledge Base**
III: Search for Existing Market as Template
- **Market Repository**
- **Market Templates**
Few Hours – Several Days
Initial Requirements
Development Phase
Repeat
IV: Prioritize Requirements
- Taskboard
- Story Cards
V: Concretize, Estimate & Choose Requirements
- Planning Poker
- Taskboard
- Story Cards
VI: Design, Implement & Test
- Automated Testing
- Cont. Integration
- Burndown Charts
- **Experiment Center**
Timeboxed: 1-4 Weeks
New Market Releases
New Requirements
Operation Phase
VII: Release & Deploy Market Platform
- Automated Deployment
- Automated Dependency Resolution
- Automated DB Migration
VIII: Observe & Assess Market Activites
- Logging Framework
- **Historic Time Series Data Store**
IX: Add & Revise Requirements
- Taskboard
- Story Cards
- **Market Design Knowledge Base**
- **Historic Market Data Store**
Continuously
Legend:
Step X
Single Process Step
Software Artifacts
Regular Third-Party artifacts
Bold Artifacts developed as part of this thesis project

Fig. 4.1. The Agile Market Engineering Process Model

process steps, user roles, actions, or software artifacts the whole framework merely aims at providing a coherent collection of best practices, experiences, and tools, and furthermore shows one way to orchestrate them, which proved to be successful in previously accomplished market engineering projects. In this spirit, the following sections describe the core *“building blocks”* of agile market engineering that have been tested and were found to be useful in the past.

In Section 4.1 a lightweight and agile market engineering process is described. Section 4.2 then provides an overview of specific software artifacts that can support this process.

Several general terms and notions are frequently used throughout the following process descriptions, which generally follow the common terms and notions of Scrum (see also Section 2.1.2). Requirements are collected in a list called **product backlog**. It consists of a set of **user stories** where each story describes a small piece of required functionality on an abstract level (i.e. detailing *what* functionality is required but not *how* it should be realized). Stories are usually written down on story cards as shown in Figure 4.2. A user story can be broken down into several specific *tasks* that need to be accomplished for the user story to be completed.

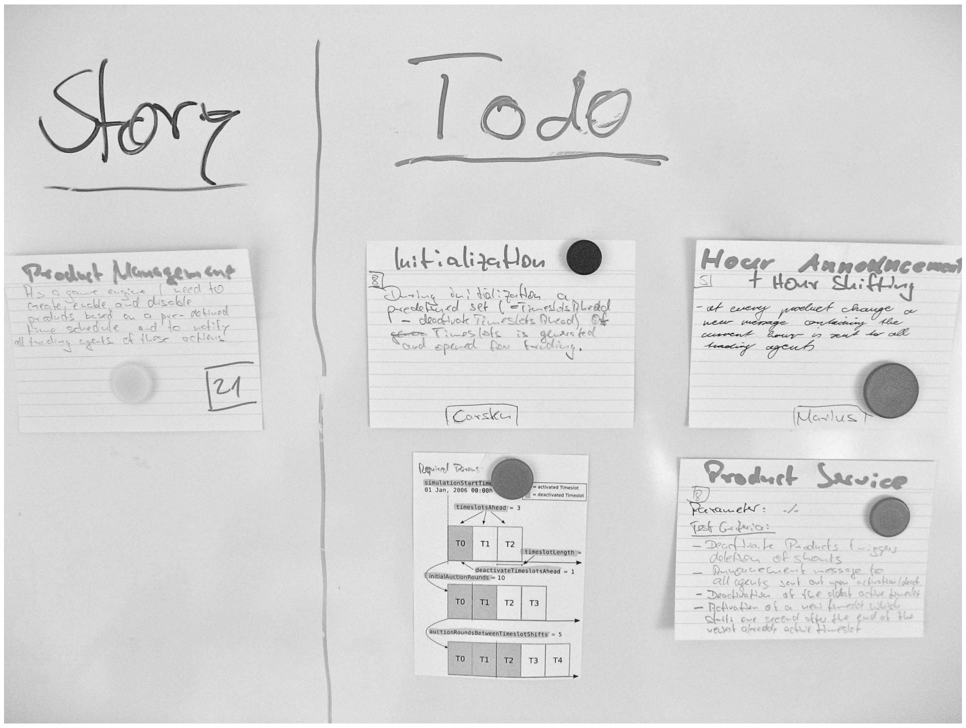


Fig. 4.2. A sample TAC Energy User Story broken down into several tasks

Technically, the product backlog is maintained on a *Taskboard*. In its simplest (but usually sufficient) form this is just a magnetic whiteboard where the user stories and their corresponding tasks can be easily pinned up and sorted manually with minimal effort required. Figure 4.3 shows a stylized taskboard, Figure 4.4 shows a snapshot of the Taskboard used to track the development of the TAC Energy project.

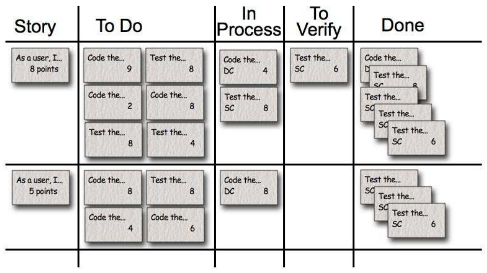


Fig. 4.3. Taskboard Mockup (Source: Cohn, 2009)

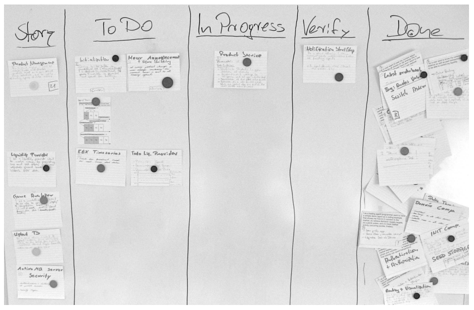


Fig. 4.4. Taskboard used in TAC Energy Project

4.1 A Lightweight and Agile Market Engineering Process Model

In general three different agile market engineering phases can be distinguished (see Figure 4.1), *"Pre-Development"*, *"Development"*, and *"Operations"*. As already described in Section 3.3.2 the pre-development phase roughly comprises market assessment and theoretic market modeling activities as covered by the frameworks of Kambil and van Heck (2002) and Neumann (2007). The development phase comprises all tasks related to the technical creation and implementation of the new market platform, while all tasks pertaining to running the market are subsumed in the operation phase.

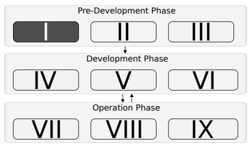
The *Pre-Development Phase* occurs only once and is dedicated to developing the overall market idea and the attached business model. *Development* phases are repeated in short cycles, each delivering potentially shippable and incrementally improved versions of the market platform. The *operation phase* continues to take place over time and in parallel to the development phases. From market operation new ideas on how to improve the platform arise based on observing market activity and are mirrored back to the development phase. Each of these phases is described in more detail in the following sections.

4.1.1 Pre-Development Phase

During the pre-development phase an initial concept for the future (electronic) market is developed. This concept includes the development of a broad vision for the new market, a thorough analysis of the existing market environment, and an assessment of previously developed market platforms in search of a sufficiently similar market as template to build the new project upon.

Develop Market Vision and Assess Market Environment

Phase:	Pre-Development Phase
Responsible:	Business Owner
Also involved:	—



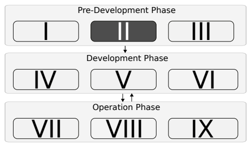
Finding and formulating a vision is a creative process. Different techniques can be used to foster creativity. One of them is called *freewriting.* Belanoff et al. (1991) provide an inverse definition for this technique: *"Freewriting is what you get when you remove almost all of the normal constraints involved in writing."* This means that a person (in the market engineering context the business owner) sits down, takes a blank sheet of paper, ensures that there will be no interruption for the next 10 minutes, and starts writing whatever comes to his mind after (initially) thinking about the future market to be developed. The only strict rule is to continuously put down words on the paper without stopping or correcting anything. Spelling, grammar, topic, line of argumentation, quality, correctness – all this is completely unimportant as the outcome will never be exposed to to anybody else.

(Elbow, 1998, p.9) notes on freewriting that *"much or most of it will be far inferior to what you can produce through care and rewriting. But the good bits will be much better than anything else you can produce by any other method."* And exactly these *good bits* will be the basis for the vision and the concept of the new market.

Once the business owner has developed a sufficiently concrete vision by repeating the freewriting technique (or other techniques such as brainstorming or mind mapping) the process / stakeholder benefit framework of Kambil and van Heck (2002) can then be used to systematically explore the market environment and to identify important stakeholders. Also, potential obstacles and opportunities for the new market idea can be assessed using their methodology so that in the end, a strategy on how to position and introduce the new market into the existing environment is clearly defined. Further information on how to apply this framework is provided in Section 2.2.1 an by the authors in Kambil and van Heck (1998, 2002).

Collect Initial Requirements

Phase:	Pre-Development Phase
Responsible:	Business Owner
Also involved:	Market Expert



After the market vision is formulated and the market environment is assessed thoroughly initial requirements in form of "user stories" (Jeffries, 2001) each describing a small facet of the required overall functionality are collected and written down by the business owner who is supported where necessary by the market expert. The collection of all user stories is called *product backlog*. According to Cohn (2004) a user story should be . . .

- independent
- negotiable
- valuable to end users (market participants)
- estimable
- small
- testable

Wirdemann (2009) suggests to write user stories from an end user perspective and in the language of the end user detailing the requirement (the *"What?"*) but leaving out implementation details (i.e. the *"How?"*) completely. The rationale behind this is the observation that the requirement itself doesn't change very often while the implementation part does as the project goes on an insights from implementing the overall market are collected. Separating the requirement part from the implementation part saves the business owner from rewriting user stories whenever new ideas and solution approaches on how to best implement them are found or developed.

Consequently, Cockburn (2008) calls a user story *"the title of one scenario"*, which inherently bears *"a promise for a conversation [...] about the scenario for which this is the title!"*. At this point in the overall process it is sufficient to find the "titles" that basically serve as mnemonics for conversations to be conducted later on (during the development phase) between business owner and market developer on *how* to realize the story.

As a good practice for writing user stories the following textual format has been proposed: *"As a [User Role] I need a [functionality] so that I get [business value]"* (Cohn, 2004; Gloger, 2008).

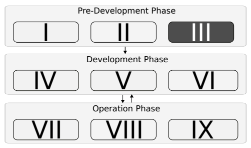
Thus, examples for user stories would be something like "As a market participant I need to be able to list all my orders so that I can track their statuses and don't lose the overview." or "As a business owner I need to be able to lock, unlock and delete user accounts so that I can appropriately react to misconduct and violations of market rules."

Technically, writing and collecting user stories on small (A5 size) index cards was found to be quick, simple, persistent, easy to appended (and to dispose) and also allows for simple sorting and reordering on a whiteboard or a flip chart as shown in Figure 4.4. This board is also called *taskboard* and it fosters transparency and collaboration among all market developers and business owner. Story cards can be used later on to protocol the conversation between business owner and market developer on how the particular user story should be implemented.

Specifically in the context of agile market engineering, the business owner (possibly supported by the market expert) also needs to ensure that the market micro structure, i.e. the market rules that define allocation and pricing rules, the market language, and information dissemination policies (Anandalingam et al., 2005, see:) are set according to sound economic principles. Here, existing experiences and knowledge from previous market engineering projects and from mechanism design research need to be taken into account. For this purpose a market design knowledge base was developed that is able to store market design recommendations in a flexible parameterized format and makes them easily accessible and searchable (Block and Neumann, 2008). Section 4.2.1 provides more details on this agile market engineering software artifact and its usage.

Search for Existing Market as Template

Phase:	Pre-Development Phase
Responsible:	Business Owner
Also involved:	Market Expert, Market Developer



With the vision, an initial set of user stories, and a sound mechanism design at hand the BO possesses an abstract description of the new market. Supported by ME and MD the BO can now examine previously developed market platforms that can possibly serve as a technical basis for the development of the new market. For this purpose a specific market template repository was developed that is capable of storing, maintaining, and easily provisioning (the source code of) different market instances (Schönfeld and Block, 2010). Initially, two simple but fully functional market instances (a call market and a continuous double auction market) were developed and contributed as "initial endowment" to the template repository. In the mean time several market platforms originate from these templates (see Chapter 5) and a specific prediction market template has been contributed back to the repository as new for future prediction market projects to build on.

Though several different market instances are available as templates, it is well possible that the product owner does not find any them particularly well fitting for his new market idea. In this case the development process begins without a template and is therefore somewhat slower initially.

If a template is found, it can be instantiated with minimal effort[1] and run out of the box on every computer that has the required development environment installed. This allows the business owner – possibly supported by the market expert – to test and evaluate the different template market platforms "live", in order to find out if

[1] "Minimal" in this context means that – provided the base software development framework is installed on the system – instantiating and running the market template can be accomplished with three command line statements and should take no longer than approximately a minute depending on how fast required and automatically installed third-party artifacts can be downloaded from the Internet (see also Section 4.2.2).

one of them is similar enough to the system he envisions and thus can be used as a foundation to build on.

Even if no template sufficiently fits to the envisioned new market the process of inspecting and "hands-on" testing existing templates helps the business owner to better express his requirements by pointing to existing functionality in the templates and by delineating that from functionality required for the new market platform.

The technical functionality of the market repository as well as its usage is described in more detail in Section 4.2.2. Design, implementation and core functionalities of the initial market templates in the repository are described in more detail in Section 4.2.2.

4.1.2 Development Phase

Software development (and as such the development of electronic markets) happens within given boundaries for cost, time and scope. Consequently unforeseeable or unplanned changes during project runtime and erroneous plans or estimations lead to situations where tradeoffs between these three constraints have to be made. Three core principles guide the development phase of the agile market engineering process and prescribe resolution strategies for these situations:

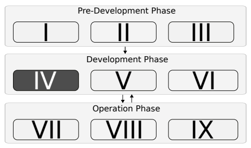
Self Organization: The Market developers take over sole responsibility for the development process and for organizing their work. Following this principle requires some learning in the beginning of a project where errors are made (e.g. overly optimistic estimations) and tasks are not executed completely. Especially in this phase, a *change manager* (see Section 3.3.1) can provide valuable support to MD and BO by providing them with an an outside view on their work helping them to better organize themselves and their work processes without prescribing or enforcing solution concepts or process workflows.

Timeboxing: Project meeting durations and deadlines are defined upfront and are strictly enforced by the change manager. This enforcement forces all involved persons to work concentrated and focused on the tasks of the respective meetings. Also, keeping time constant, means that a development cycle is never prolongated even if e.g. requirement changes or obstacles in the implementation process occur. Instead the amount of delivered functionality has to be adjusted.

Pull-Principle: The pull-principle is based on the Kanban idea (Louis, 2006) and regulates the number of user stories chosen for implementation in a specific development phase. The basic idea is to let the market developers (and not the BO) chose ("pull") exactly that number of user stories from the product backlog they think can be implemented in one single development phase. Like this a self regulatory circuit is created that ensures that the MDs work capacity is used but also prevents overload.

Prioritize Requirements

Phase:	Development Phase
Responsible:	Business Owner
Also involved:	—



In this step all previously collected requirements (in the form of user stories) are prioritized by the business owner. The simple base rule for prioritization is that most important stories are put to the top of the product backlog, not so important requirements to the bottom. Still, the question remains what *important* means. To answer the question Clegg and Barker (1994) and Ash (2007) developed the MoSCoW Prioritization scheme. MoSCoW stands for

Must Have (M): key requirements that have to be implemented in any case before the market could be released

Should Have (S): equally important requirements as must haves but not necessarily to be delivered within the current timebox as there is an acceptable workaround or alternative that can be used until the requirement is implemented

Could Have (C): "nice-to-haves" that add value to the market from a customer point of view but that can be dropped or retarded to a later timebox if the implementation effort was initially underestimated

Won't Have (W): All requirements that are on the original list of the business owner but that won't be delivered within the current development cycle (timebox).

During the development cycle the 'M' stories are implemented first, followed by 'S' and 'C' stories if time permits.

A business owner who prioritizes his initial list of user stories into the above categories immediately understands what this prioritization means for the current development cycle. E.g it is immediately clear to him that 'W' stories are certainly *not* delivered in this cycle, which might be not as clear if categories such as 'high', 'medium' or 'low' are chosen as category names.

Selhorst (2006) proposes another prioritization scheme, which was originally developed by Kano et al. (1994) for the measurement of customer satisfaction. Here requirements are divided into the following categories:

Must Be (M): As in MoSCoW must be requirements have to be implemented in the initial release

Surprise and Delight (S): Are features that help differentiate the own product from that of competitors. Amazon's 1-click-Buy is an example for such a feature. Market participants did not expect this feature but they like it as it reduces their efforts (i.e. their transaction cost) for buying a good to almost zero.

Better Not (N): Requirements that fall into this category have to be rethought because market participants will likely object to using the platform due to these features. An example could be a forced registration before any market functionality can be used though, for pure browsing and searching of the available products in the market, such a registration would not be necessary.

More is better (M): These requirements are also called linear requirements. More of it is better. The time required for order execution at a stock exchange is an example for such a requirement: The shorter the better. On the other hand, more of these requirements is usually also more expensive to build. Thus a balance between expected increase in customer satisfaction and expected increase in development cost has to be found.

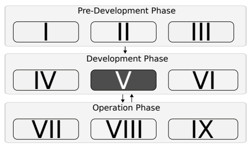
Indifferent (I): These requirements do not affect the acceptance ratings of market participants. An example could be the back-end software used to operate the market. For the market participant it is of no importance if this technology is based on a PHP or a Java server software as long as it delivers the value-add the customer expects.

For the assignment of requirements into the above categories Selhorst recommend collecting preferences from real (potential) customers by means of questionnaires. The collected feedback is then used as the basis for prioritization.

Either way, at the end of this process step all user stories in the product backlog are prioritized according to business value, with important stories first.

Concretize, Estimate & Choose Requirements

Phase:	Development Phase
Responsible:	Business Owner, Market Developer
Also involved:	—



Concretize requirements: Essentially, each user story is a short, high level description of an idea or a requirement. Cohn (2004) calls them *"promises for a conversation"*. Now these conversations need to take place. Market developer(s) and business owner step through the prioritized product backlog discussing each of the stories in detail. The purpose of this exercise is to build a common and detailed understanding between business owner and market developer(s) on what is takes to realize each of the stories. A short transcript of the conversation can be written down on the story card itself together with testable acceptance criteria. Specific tasks that belong to a user story can be written on additional cards as shown in Figure 4.3 and 4.4. These are used later on to judge if a particular user story is "fully implemented", which is the case only if it fulfills all of the listed acceptance criteria.

Estimate stories by size: With a detailed understanding on what it takes for each user story to be completely realized and tested, the market developers then estimate the size (and *not* the effort) of each story.

The rationale for estimating user story sizes and not development effort is the following: Effort basically measures the time duration required for accomplishing a certain task. Estimating efforts in an environment of industrial piecework manufacturing is likely to yield quite precise results as each task is well defined or can be broken down into well defined subtasks each being measurable in *time required for accomplishing the task*. For example a task of "assembling workpiece 1 and workpiece 2" can be broken down into the subtasks *"reach for workpiece 1"* (effort: 2 seconds), *"reach for a screw"* (effort: 2 seconds), *"reach for workpiece 2"* (effort: 3 seconds), *"screw together workpiece 1 and workpiece2"* (effort: 8 seconds). The overall effort for accomplishing the task is thus estimated at 15 seconds overall. This estimate is assumed to be (almost) independent from the person or team who accomplishes the task.

For software projects this approach is likely to produce much more imprecise results. First, writing a software artifact is usually a unique task that has not been accomplished before and thus requires a certain amount of (difficult to predict) creativity and development experience; empirical evidence suggests that very good software developers can be 25 (Spolsky, 2005) to 37 (Coplien, 1994) times more productive than average ones. Consequently, estimating effort (i.e. time duration) for the realization of a software development task is strongly dependent on the particular person or team executing the task. But this in turn is usually not determined yet at the time of estimation. For these reasons estimating effort is not recommended (Gloger, 2009).

Still predictions are important for a project's time and resource management. Cohn (2005) thus proposes to estimate the *size* of user stories instead of the required efforts as follows: Out of the list of all user stories a small and well understood story is chosen as reference and assigned an (arbitrarily chosen) *"size"* value.[2]. Already the process of choosing a reference story and assigning it a size value forces market developers and business owner to discuss (and define for their project) characteristics and factors that essentially make up the size of the story. These factors may include the story's perceived "complexity", the underlying technology used for its realization, etc.

With a reference story at hand the market developers can start to estimate the sizes of all other stories as multiples of the reference story's size. For example, if the reference story is assigned a value of 3 and the developers perceive another story as being twice as large, they assign that story a size of 6. Cohn (2005) and Gloger (2009) suggest to use only few discrete values for story sizes that follow a stylized Fibonacci sequence as shown in Table 4.1.

Step	0	1	2	3	4	5	6	7	8	9	
Estimated Story Size	0	1	2	3	5	8	13	20	40	100	?
Estimation Tolerance +/-	0	0.5	1	1.5	2.5	4	6.5	10	20	50	?

Table 4.1. Stylized Fibonacci Series (Source: Gloger, 2009, p. 141)

[2] *"Size"* in this context is basically a unit-less measure though the measurement unit name *story points* is oftentimes used in the agile software engineering community.

This approach prevents the developers from discussing if a story has a size of – say – 48 or 49, which would be not too meaningful a difference. Additionally stories estimated at 40 or 100 story points are implicitly also marked as being "difficult to estimate" while stories with sizes of 3 or 5 are not only small but also well understood and well predictable. 40 or 100 point stories likely have to be specified in more detail and will possibly be broken down into smaller (and thus better estimable) sub stories.

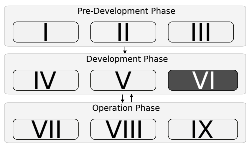
A popular technique to efficiently and quickly estimate large amounts of user stories is called *planning poker*. This method is a consensus-based estimation technique developed by Grenning (2002) and based on the Wideband Delphi method (Boehm, 1981). Details on how this method can be practically employed are provided by Cohn (2005); Gloger (2009); Wirdemann (2009).

Choose accomplishable subset of stories for current development phase: After all user stories are estimated the market developers need to select *that* number of the topmost user stories from the product backlog *they* think can be *completely* realized in one single development phase (pull-principle, see above).

It should be noted that the initial estimations in the first few development cycles will likely be imprecise. But with each new development cycle the market developers get a better feeling for their work capacity, which is sometimes also called *team velocity* of the MDs. With this knowledge at hand they will be able deliver more and more precise estimations over time. Knowing the velocity of the market developers helps the business owner to better plan ahead. Based on the velocity, i.e. the number of story points the market developers are able to deliver during one development cycle (on average), and with the estimated sizes of the stories in the product backlog he will be able to predict how long it will take until certain user stories are implemented. Based on this knowledge, he will finally also be able to estimate cost for the development of a specific story (if required).

Design, Implement & Test

Phase:	Development Phase
Responsible:	Market Developer
Also involved:	Business Owner, Market Expert, Change Manager



The most important thing to note about the implementation and testing step itself is that it is supposed to last only a *short* period of time. If one single cycle takes more than a few weeks (at most) something has very likely gone wrong.

With a detailed work plan at hand the technical development process itself can start. Ideally all market developers sit in one large room with one or more separate meeting rooms available nearby for bilateral discussions. Like this informal (or in the nomenclature of Cockburn *osmotic*, see Section 2.1.2) communication among the market developers becomes possible. Also, business owner, market expert and change manager should be in close physical vicinity to the market developers so that questions pertaining to market design, to realization details of certain user stories, or to impediments relating to team or process organization can be quickly addressed by the respective experts without any formal meeting appointments.

One story at a time: Members of the market developer team then start developing by shifting the topmost user story on the task board into the WiP (work in progress) column. Ideally they will all focus on one story only at a time so that at the end of the current development phase at most one story is in WiP state. Sometimes this course of action is impractical as the story might be too small to be shared among several market developers. In this case, some parallel work on separate stories is appropriate though the stories should be independent from each other in order to avoid situations where one market developer is blocked in his work progress by another developer who has not yet finished a dependent task required as prerequisite.

Test Driven Development: Production code for the new market is preferably written in a test-driven manner. This means that a market developer first writes several unit and integration tests that cover typical use cases as well as cases with extreme, invalid, or missing data. For each of these cases expectations on how the production software should behave (e.g. delivering a particular value as result of a computation or throwing a particular error if invalid data was supplied) are defined. All of these tests will initially fail when being executed as no production code is written yet. But with this testing harness in place, the market developer can then start writing the production code, which – in the end – has to be executed against all previously developed test cases without any of them failing. A more detailed description of the test-driven development methodology including many examples is provided by Beck (2002).

Additionally, to ensure that all features and processes are fully operational, automated user acceptance testing software such as Selenium,[3] WebTest,[4] or Watij,[5] can be used. These tools allow the definition of arbitrary task sequences that can later be automatically executed in a web browser. For example, the code in Listing 4.1

Listing 4.1. Web Browser Task Sequence for Offer Submission (see: Block and Chen, 2007)

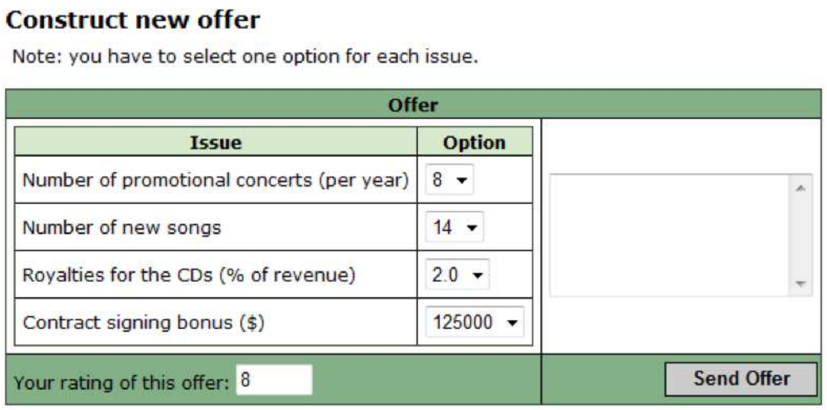
```
//Login
IE ie = new IE("http://invite.concordia.ca/invite");
ie.textField(id("username").set("MyUserName");
ie.textField(id("password").set("secret_password");
ie.button(name("submit").click();

//Submit order
ie.link(id("Send offer")).click();
ie.textField(id("message")).set('New offer...');
ie.selectList(id('concerts')).option(8).select();
ie.selectList(id('new_songs')).option(14).select();
ie.selectList(id('royalties')).option(2.0).select();
ie.selectList(id('bonus')).option(125000).select();
ie.button(id("submit_offer")).click();

//Check response
assert(ie.tag("title").value() == "Offer Submitted!")
```

[3] http://seleniumhq.org/

[4] http://webtest.canoo.com

[5] http://watij.com/

defines a task sequence that (i) logs in to the electronic market system, (ii) browses to the offer submission form, (iii) submits an offer to the market system by automatically filling in the appropriate web form (see Figure 4.5) with pre-defined values, and (iv) checks that the returned page is named “Offer Submitted!”. Task sequences


Construct new offer

Note: you have to select one option for each issue.

Issue	Option
Number of promotional concerts (per year)	8
Number of new songs	14
Royalties for the CDs (% of revenue)	2.0
Contract signing bonus ($)	125000

Your rating of this offer: 8

Fig. 4.5. Invite order submission form (Source: Block and Chen, 2007)

like this can be define for all use cases a market participant is supposed to be able to execute on the developed platform and thus allow for a close inspection of the operativeness of all desired market features from an end users point of view (Block and Chen, 2007).

Definition of Done: Once the production code for a user story is written and executable against the respective unit and integration tests, the question arises on when the user story is *done*. At the end of the development phase the business owner reviews all implemented stories and has the privilege of finally judging them “done” or “not done”. But, in order to improve clear judgement criteria and concrete guidance to MD throughout the development phase a written and commonly accepted *definition of done* (DoD) should be jointly developed by business owner and market developers. Appendix C contains a sample DoD developed and used for the TAC Energy project (which is described in more detail in Section 5.6).

Source Code Management & Continuous Integration: Every change in production and test code should be checked in to a source code management system (SCM), which provides a change history allowing users to easily revert erroneous code additions, and also ensures that all market developers share one single latest code version. The SCM system should also be complemented by a continuous integration system (CI Server) that monitors the SCM repository for incoming code changes and then (i) executes all unit and integration tests against the new version of the software and (ii) automatically compiles a new binary release of the (market platform) software. Errors during test execution or during compilation of the binary release can then immediately be reported back to the market developers so that they get quick feedback on how their code changes affect the overall system. For example, in the TAC Energy project the open source systems Hudson[6] for continuous integration in combination with Bazaar[7] for SCM are used.

Continuous Status Reporting: After a story is finished by a market developer he shifts the corresponding story card from the *WiP* column on the taskboard into the *Done* column and then updates the burndown chart by drawing a new point on it using the current time on the x-axis and the previous story point value minus the story points of the story just finished as the y-axis value. Figure 2.5 show a sample burndown chart from a TAC Energy coding week. This chart is simple to maintain and to update on the one hand, and on the other hand immediately visualizes the current progress of the respective development phase to MDs, BO and interested others.

Retrospective: At the end of each day a short 10-15 minute introspection meeting is held. Moderated by the change manager each market developer reports on (i) what he achieved that day, (ii) what he wants to achieve the next day, and (iii) which problems or impediments delay his progress. The change manager will take any reported problem on his agenda trying to find solutions for it as quickly as possible in order to provide the MDs with an optimal development environment.

Review: At the end of a development phase a review meeting is held. First the objective(s) of the current development phase and the selected product backlog is presented again by the business owner. Then the market developers present and describe all user stories they completed according to the DoD. The product owner (potentially supported by beta testing market participants) then gets a live demonstration of the newly implemented features on a *running* testing system. The product owner now has the final say in judging each of the the implemented stories as *done* or *not donqe*.

All stories that are not considered done as well as those stories that were not finished during the current development phase are returned to the product backlog and then re-prioritized at the beginning of the next development phase.

Market Simulations: Sometimes, especially during the introduction of new market mechanisms, the expected dynamics of the new market are not completely foreseeable for the business owner and the market expert. In these cases the market features are done in a technical sense but need further acceptance testing, e.g. through simulated market participants that expose a behavior similar to that of expected real market participants. In these cases a market simulation environment developed particularly for this case might proof to be useful. Instead of a deployment to the production systems the market can be deployed to the *market simulation framework* instead, which is an agent based simulation environment developed specifically for this purpose. Once the market is installed in the market simulation framework, various types of electronic agents can be defined that subsequently simulate market participants and their trading behavior on the market. Section 4.2.3 describes the market simulation framework in more detail.

Release: After the review meeting is finished and no specific market simulation runs are required the current development phase ends. On the one hand this event

[6] `http://hudson-ci.org/`
[7] `http://bazaar.canonical.com`

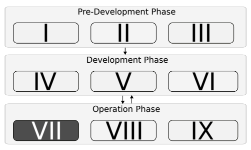
triggers the start of a new development phase, that receives all unfinished user stories as initial input, on the other hand a deployment of the latest release of the market platform to the production systems is triggered. This latter process is described in the following section.

4.1.3 Operation Phase

Contrary to the development phase, which is clearly timeboxed and repeatedly executed, the operation phase is a continuous process. It starts with the initial deployment of the market system to the production environment and then continues without interruptions in parallel to further development phases until the market platform is forcibly stopped for some reasons (e.g. bankruptcy of the business owner). This section describes (i) a best practice release process for the market as the connecting element between development and operation phase, (ii) several ways to observe and assess market activity, and (iii) possibilities to collect explicit and implicit feedback from market participants and other stakeholders in order to add or revise requirements (user stories) for further market development phases.

Release & Deploy Market Platform

Phase:	Operation Phase
Responsible:	Market Developer, Market Operator
Also involved:	—



The preceding development phase ends with a review meeting. In this meeting the newly implemented features are either approved or rejected by the business owner. In case of an approval, the new, feature-enriched version of the market platform has to be released and deployed to the production market server environment. This process is described in more detail in this section.

The release and deploy process starts with the creation of a code snapshot. The goal of this step is to create and archive the latest version of the platform's code base in an archivable form for later reference and for possible rollbacks if later versions of the platform should cause problems. At first, in the source code management system, the market developer assigns the code base a version tag (e.g. `v0.1`) so that this particular state can easily be recovered later on. Next he checks out the source code and compresses it into an archive file named after the current version (e.g. `market_v0.1_src.zip`). The source code should also be compiled and the resulting binary code should be archived into a separate file (e.g. `market_v0.1_bin.zip`). The market developer (i) keeps these files in a separate version history on a file server and (ii) hands over the binary version of the code to the market operator for deployment.

The market operator then starts deploying the new binary release of the market platform to a so called *staging system*. The staging system is (ideally) an exact mirror of the software and hardware infrastructure of the production system, and thus serves as a realistic testing environment. Any errors during the deployment of the market platform to the staging system will cause the deployment process to halt. In that case the market operator notifies the market developer who then analyzes the causes of the error(s). If these cannot easily be fixed (e.g. by correcting a simple mis-configuration) the deployment process is cancelled, the current version of the market platform is kept operational, and the market developers start the next development phase with a the particular bug fixing task as highest priority item in the product backlog.

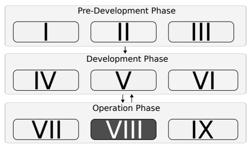
If the deployment of the new version of the market platform to the staging system was successful, business owner and market developers test the proper functioning of the running system.

Once all testing activities on the staging system are successfully completed, the market operator takes the production system offline so that a complete and consistent backup of all its data can be generated. Afterwards he deploys the new release of the market platform to the production environment and ensures that the market (as well as all dependent systems such as mail server and database) are running stable and without errors before making the new release of the market platform available to the open public. In case any errors occur in this step an immediate rollback to the previous market version and the saved market data is executed and developers are requested to take care of the errors in the next development phase.

Otherwise the market operator starts observing market platform log files and server load ensuring stable operation of the system as described in more detail in the following section.

Observe & Assess Market Activity

Phase:	Operation Phase
Responsible:	Market Operator, Business Owner
Also involved:	Market Expert



Market activity needs to be monitored on two different levels, (i) business level and (ii) technical level. On the business level the business owner needs to ensure the integrity of the market activity. This includes verification, authentication, and authorization of market participants but also the detection and prevention of fraud or other forms of market mis-conduct. One approach to accomplishing this task was developed by Blume et al. (2008). They propose a graph-based approach for fraud detection and market activity monitoring that was used to supervise the Stoccer market and is now also used to monitor the EIX market (see Section 5.4) in order to detect fraudulent user behavior on the markets. Therefore regular trade and quote data (TAQ) is represented in a graph structure and analyzed e.g. to detect cycles, which could be an indication for money laundering or – in the case of prediction markets – for fraudulent money shifting between fake and real user accounts.

Sometimes, besides TAQ data even more information about user behavior need to be collected, e.g. to better understand how market participants interact with the market platform's web interfaces so that (web) design weaknesses can be detected and addressed. In these cases, all markets based that are based on the default market templates (described in more detail in Section 4.2.2) can easily be equipped with extra data recording plugins for click-stream [8] and audit-logging. The first plugin records each single click of a user in his web browser providing the business owner with a complete (but also complex) picture on how market participants use and navigate the market platform. The second plugin records each database transaction with timestamp and the respective market participant's name in a separate database table effectively allowing the business owner to reconstruct *all* data manipulations on the market platform in their original chronological order.

On the technical level, the market operator needs to monitor resource utilization of the production system environment as well as runtime errors of the market platform itself in order to adjust the technical runtime environment to changing usage patterns and to appropriately dispatch runtime error messages to market developers.

For server monitoring specialized software such as OpenNMS [9], Zenoss [10], or PandoraFMS [11] can be used. The monitoring of runtime errors can be accomplished through the logging framework already included in the market templates (see Section 4.2.2). This framework allows for a flexible configuration of logging targets [12] as well as fine-grained log levels[13] that can be configured separately for different parts of the application. For example, all log messages at minimum log level 'WARN' from the market's settlement component can be sent out via email while log messages from the web interface are only logged if they are of type 'ERROR' and will then be written to a special log file. The default market templates come with a dynamic logging plugin installed that allows the market operator to adjust log levels of the different market components during market runtime via a web interface shown in Figure 4.6.

[8] `http://www.opensymphony.com/clickstream/`

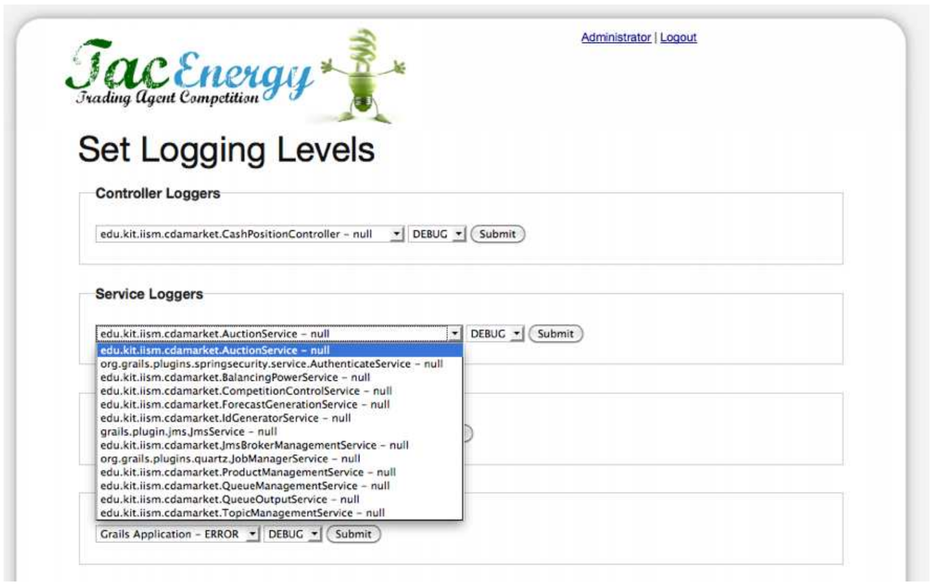
[9] `http://www.opennms.org`

[10] `http://www.zenoss.com`

[11] `http://pandorafms.org`

[12] Logging targets are output devices including files, database tables, e-mails, or short messages (SMS) where logging information can be written to.

[13] A log level defines a minimum threshold level, such as 'INFO', 'WARN', or 'ERROR'. Each log messages is assigned one of these levels. During runtime the MO can adjust the market system's log level effectively filtering out all log messages below this level.


Administrator | Logout

TacEnergy Trading Agent Competition

Set Logging Levels

Controller Loggers

edu.kit.iism.cdamarket.CashPositionController – null | DEBUG | Submit

Service Loggers

edu.kit.iism.cdamarket.AuctionService – null | DEBUG | Submit

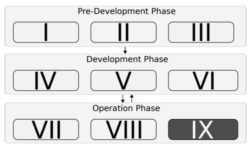
```
edu.kit.iism.cdamarket.AuctionService – null
org.grails.plugins.springsecurity.service.AuthenticateService – null
edu.kit.iism.cdamarket.BalancingPowerService – null
edu.kit.iism.cdamarket.CompetitionControlService – null
edu.kit.iism.cdamarket.ForecastGenerationService – null
edu.kit.iism.cdamarket.IdGeneratorService – null
grails.plugin.jms.JmsService – null
edu.kit.iism.cdamarket.JmsBrokerManagementService – null
org.grails.plugins.quartz.JobManagerService – null
edu.kit.iism.cdamarket.ProductManagementService – null
edu.kit.iism.cdamarket.QueueManagementService – null
edu.kit.iism.cdamarket.QueueOutputService – null
edu.kit.iism.cdamarket.TopicManagementService – null
```

Grails Application – ERROR | DEBUG | Submit

Fig. 4.6. Web Interface for dynamic adjustment of log levels in TAC Energy

Add & Revise Requirements

Phase:	Operation Phase
Responsible:	Business Owner
Also involved:	Market Expert, Market Developer, Market Operator

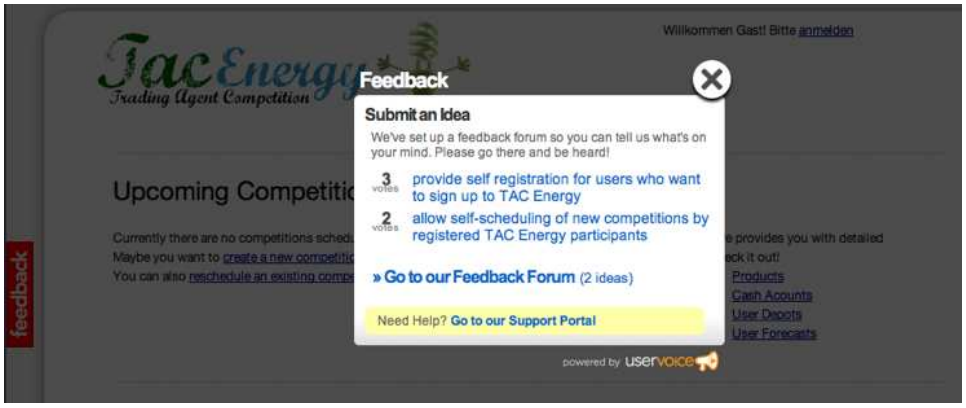


The running market system is a rich source for the elicitation of feedback on how to make the market even more convenient and efficient to use.

Two different forms of feedback can be distinguished, explicit and implicit feedback. Explicit feedback methods require extra efforts from users. The might, for example, require users to answer questionnaires or to accomplish interviews. Implicit feedback methods are more unobtrusive in that the natural interaction of users with a system is observed and evaluated (Kelly and Teevan, 2003).

In information systems literature questionnaire based (i.e. explicit) models for feedback elicitation and evaluation are well established and have a long research tradition. For example, models aimed at testing user acceptance of specific information systems like the Technology Acceptance Model (TAM, see Davis et al., 1989; Venkatesh et al., 2003) have attracted much research for over 20 years though recently the TAM model and its derivatives have been criticized for their *"questionable heuristic value, limited explanatory and predictive power, [...] and lack of any practical value"* (Chuttur, 2009).

Besides rather complex and difficult to evaluate TAM style models for feedback elicitation and evaluation, very simple online feedback tools that are easy to integrate into web-applications, have recently gained much interest (Hrastinski et al., 2010). Figure 4.7 shows a screenshot of the user feedback elicitation function built into the TAC Energy project. The tool is based on UserVoice[14] online service and relies on a simple web interfaces for end users that allows them to report and rank ideas and issues that came to their attention during usage of the platform.



Feedback

Submit an Idea

We've set up a feedback forum so you can tell us what's on your mind. Please go there and be heard!

3 votes provide self registration for users who want to sign up to TAC Energy

2 votes allow self-scheduling of new competitions by registered TAC Energy participants

» Go to our Feedback Forum (2 ideas)

Need Help? Go to our Support Portal

Currently there are no competitions sched
Maybe you want to create a new competiti
You can also reschedule an existing comp

e provides you with detailed
eck it out!

Fig. 4.7. TAC Energy online user feedback elicitation tool

An implicit feedback evaluation approach is described by Vandenpoel and Buckinx (2005) who use click-stream data (see Section 4.1.3) to evaluate user behavior and preferences based on detailed recordings of their respective system interaction behavior using datamining techniques. Choros and Muskala (2009) and Schüller and Wörndl (2008) use the same source of data to visualize user clicks in called heat maps as shown in Figure 4.8. Like this interesting (i.e. often clicked-on) parts of a web page can be easily identified. This knowledge can then for example be used to to place important information in these often clicked page areas.

In short, several different techniques for implicit and explicit feedback elicitation exist and can be used as input for a continuous improvement of the market platform. Additionally, these insights should be persisted in the marked design knowledge base (see Section 4.2.1) for future re-use.

4.1.4 Summary

In summary, the previous sections described a lightweight and agile market engineering process. It consists of a "pre-development", a "development", and an "operations" phase. Each of the phases consists of three process steps each. With this process model at hand, the next sections describe several different software artifacts (also highlighted in Figure 4.1 in bold) that were specifically developed to support agile market engineering teams in their daily project work.

[14] `http://www.uservoice.com`

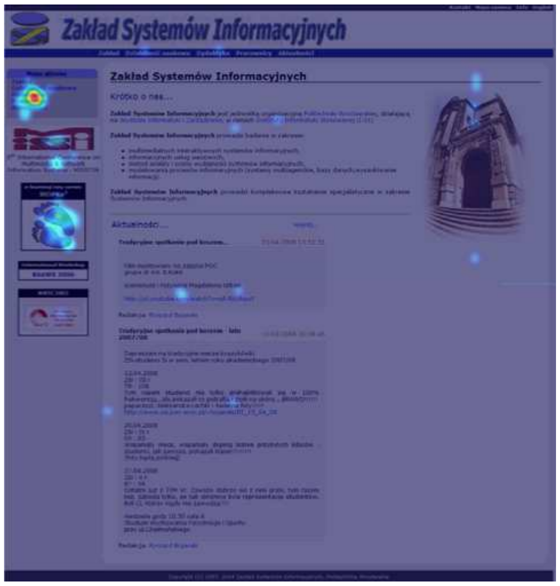

Zakład Systemów Informacyjnych
Zakład Systemów Informacyjnych
Krótko o nas...
Aktualności...

Fig. 4.8. Website Heat Map for Usability Analysis (Source: Choros and Muskala, 2009)

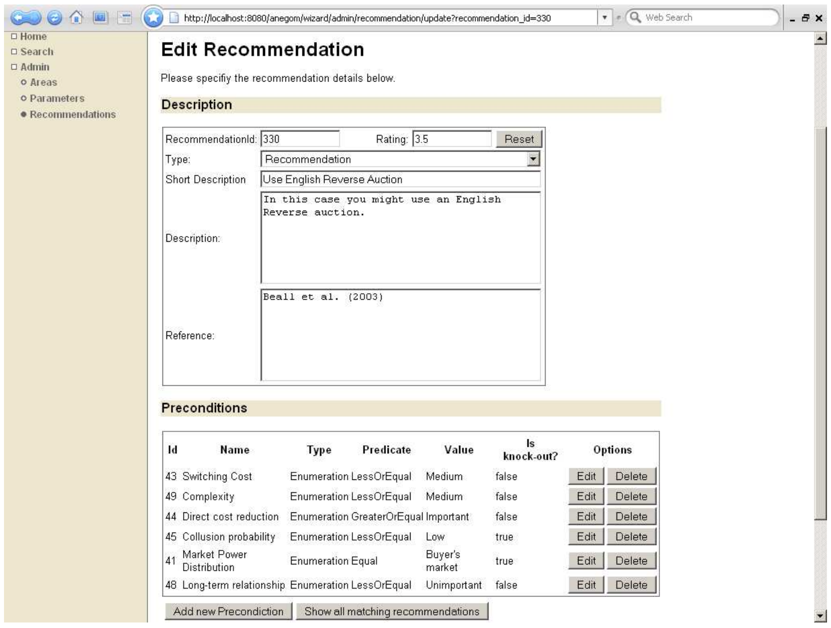
4.2 (Software) Artifacts to Support Agile Market Engineering

4.2.1 Market Design Knowledge Base

Economic research as well as field experience produced a large amount of knowledge on how or how not to design (electronic) markets covering market types as diverse as electronic catalogue systems, e-negotiations to e-auctions in its many different forms. Each situation for the application of markets is different and there exists no single best solution for setting up and running markets. Each mechanism rather has certain advantages and disadvantages. From economic theory, especially from mechanism design theory, it is well known that even small changes in the design of exchange mechanisms can have considerable impact on the outcome. Consequently, a market design knowledge base system (MDKB) was designed that is aimed at supporting business owners and market experts in their decision making process on which mechanism to choose best in a specific situation (see Step II of the agile market engineering process). Additionally, new insights from running markets can be persisted in the MDKB for future re-use in other market engineering projects.

Knowledge Acquisition, Storage and Evaluation

Davenport and Prusak (2000) define knowledge as *"a fluid mix of framed experience, contextual information, values and expert insight that provides a framework for evaluating and incorporating new experiences and information."* Thus the MDKB system has to accomplish both, providing (i) a storage facility for contextual information, values and expert insight and (ii) a mechanism that allows knowledge retrieval in a context of new experiences and information. For fulfilling the first part, the Market Design Knowledge Base (MDKB) offers several different fields to store e.g. verbal recommendations (e.g. "use an English reverse auction"), literature and other references as well as a variable number of parameters describing the preconditions for which a particular recommendation holds (e.g. "at most low probability of collusion among bidders" for the recommendation to use an English Auction). Figure 4.9 shows an example screenshot of MDKB during data entry of new knowledge.


http://localhost:8080/anegom/wizard/admin/recommendation/update?recommendation_id=330
Web Search
Home
Search
Admin
Areas
Parameters
Recommendations

Edit Recommendation

Please specifiy the recommendation details below.

Description

RecommendationId:	330
Rating:	3.5
Type:	Recommendation
Short Description	Use English Reverse Auction
Description:	In this case you might use an English Reverse auction.
Reference:	Beall et al. (2003)

Preconditions

Id	Name	Type	Predicate	Value	Is knock-out?	Options
43	Switching Cost	Enumeration	LessOrEqual	Medium	false	Edit Delete
49	Complexity	Enumeration	LessOrEqual	Medium	false	Edit Delete
44	Direct cost reduction	Enumeration	GreaterOrEqual	Important	false	Edit Delete
45	Collusion probability	Enumeration	LessOrEqual	Low	true	Edit Delete
41	Market Power Distribution	Enumeration	Equal	Buyer's market	true	Edit Delete
48	Long-term relationship	Enumeration	LessOrEqual	Unimportant	false	Edit Delete

Fig. 4.9. Screenshot of the *Edit Recommendation* Screen of MDKB

The second part is implemented by an inference mechanism that takes a set of parameters describing a situation the user seeks advice for, and computes similarities to those cases (situations) already stored in the knowledge base. Sufficiently similar cases, which consist of a set of preconditions describing the market environment as well as suitable recommendations, are returned to the user conveying knowledge on how to proceed best in the respective setting or (possibly also) on what type of market rules to avoid in that situation.

As with all knowledge based systems, the most crucial task is to acquire, adapt, verify and maintain the underlying knowledge base. Possible sources for knowledge acquisition are normative literature on auction design as well a empirical literature, structured interviews with domain experts from industry, or academic staff that focuses on mechanism design research. Also lessons learned from particular market engineering projects should be captured in dedicated retrospective sessions during the development process itself.

A specific feature of MDKB may prove to be valuable when eliciting advice during interviews: In the author's on experience domain experts were oftentimes unable to give a definite recommendation on which market mechanism to choose best for a particular domain and a specific market environment. Still they were still quite clear on which mechanism *not* to choose. E.g. in a case where strong bidder asymmetries occur one can predict in practice that an English auction will lead to an economically inefficient outcome, while it is not clear if e.g. a Dutch auction or an electronic negotiation might be the more favorable alternatives instead.[15] In such a case, the MDKB user might still receive a warning (i.e. negative recommendations) not to choose an English auction along with an appropriate explanation, which increases his awareness and helps him avoid stepping into a "trap" of severe design failures.

With a growing number of recommendations entered into the knowledge base, data consistency becomes an important issue. As users are allowed to define their own parameters, rules, and recommendations, an automated approach for consistency checks is hard to implement. Thus the current solution to this problem is to give users of MDKB the possibility to manually check at the time of entering new recommendations into the system, which other existing recommendations also match their specific set of preconditions (c.f. button "Show all matching recommendations" in Figure 4.9). Furthermore a simple five star rating system is implemented that enables users to judge recommendations and thus to provide qualitative information on the goodness of the knowledge stored.

Figure 4.9 shows a screenshot of MDKB that displays the administration page used to enter or adjust recommendations. Basically a MDKB user wishing to enter knowledge needs to specify the type of recommendation that should be stored (e.g. *"Recommendation"*, *"Warning"*, ...). Subsequently he fills in a short (and optionally an extended) description of the recommendation (e.g. *"Use English Reverse Auction"*) before (optionally) adding references to related literature or other resources that are suited to increase the credibility of the particular recommendation and may provide additional background information.

Left in this state, the recommendation would be generally valid and thus always be displayed to users of MDKB no matter which search parameters they specify. In order to limit the applicability or scope of a recommendation one can add an arbitrary number of preconditions. MDKB then only returns the recommendation as a search result if the preconditions match the search parameters.

For specifying a precondition, a MDKB user first has to choose a parameter (e.g. *"Switching Cost"* of a product) from an (extensible) list of parameters provided by MDKB. In the case of *"Switching Cost"*, the parameter is specified as an enumeration[16], which basically means that for this parameter a user defined, ordered set of parameter values is given. After having selected the parameter, the expert needs to determine the parameter value (e.g. *"Medium"*) and an operator (e.g. *"LessOrE-*

[15] The term "inefficient" is used here to express the fact that in this situation an English auction would result in a smaller expected revenue for the auction owner, than for example a Dutch auction would do. Thus, from the auction owner's perspective, the English Auction seems to be "inefficient" as compared to other alternatives.

[16] Supported parameter types in MDKB are *String*, *Boolean*, *Number*, *Decimal*, and *Enumeration*

qual"[17]) in order to finish adding the precondition. Overall, the expert specified in this case that "use English Reverse Auction" is a valid recommendation only if product switching cost are at most medium.

System Design and Implementation of MDKB

In this paragraph the implementation of MDKB is described. First, the system architecture is introduced to establish a common notion of the domain model and its interaction with the system. Subsequently, the case-based reasoning for the recommendation retrieval is shown.

Following the typical Separation of Concern (SoC) pattern, MDKB is divided into five distinct application tiers (Alur et al., 2003), user interface, controller, service, persistence and domain model. Each of these layers encapsulates its specific tasks and logic from the other layers in order to achieve a maximum code decoupling and like this a high system stability, maintainability and adaptability.

The domain model is implemented in a relational database as displayed in Figure 4.10. The main entity is called *Recommendation* which stores instances of recommendations, warnings and so on. For each recommendation to be valid, $0..*$ prerequisites must hold. These prerequisites are specific values (or value ranges) from different parameters, stored in the database. If a recommendation is true, not only a verbal description – as stored in the *Recommendation* entity – but also a structured (parameterized) recommendation stored in *Mechanism* and *MechanismParam* may be returned. These mechanism parameter sets could be parsed into several formats (XML, yaml, JSON) that afterwards might be used to automatically pre-configure market platforms.

The relational database storing the domain model is accessed from MDKB application via a distinct persistence layer, which allows the manipulation of the data using the data access objects (DAO) pattern. Like this, the underlying storage technology could be appended or switched with minimal impact on the program itself. Above the persistence layer, a service layer implements the more complex business logic like e.g. the case-based reasoning algorithm. Separating this logic from the DAO on the one hand, and from the application workflow on the other, ensures that e.g. different recommendation retrieval mechanisms could be implemented without changing the principal workflow. The last distinct layer is introduced between application workflow and view layer, the fifth layer of MDKB. This separation allows different front-ends like a HTML interface and a web services interface to be implemented transparently using the same application logic.

For the implementation of recommendation retrieval algorithms, several approaches already exist. Forgy (1982) introduces RETE, an algorithm for matching many patterns on many objects, which is oftentimes used in rule based expert systems. Many alternatives have been proposed since then, the most notable ones being TREAT (Miranker, 1987) and LEAPS (Batory, 1994). The main problem with this group of algorithms is of technical nature: Existing implementations of these rule engines rely

[17] Different operators are provided for different parameter type as e.g. LessOrEqual is not meaningful for string parameters.

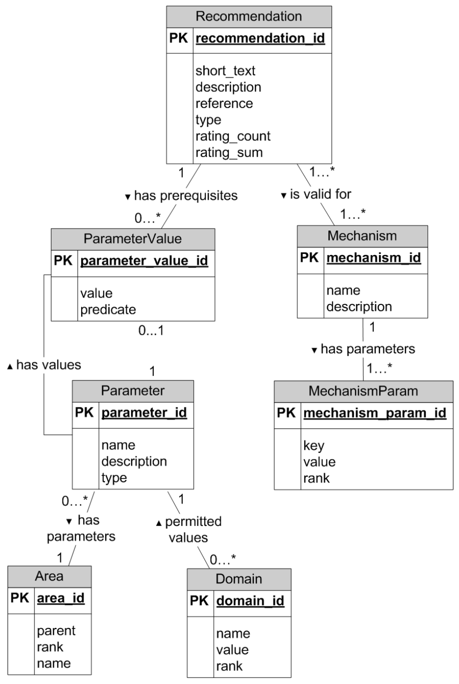


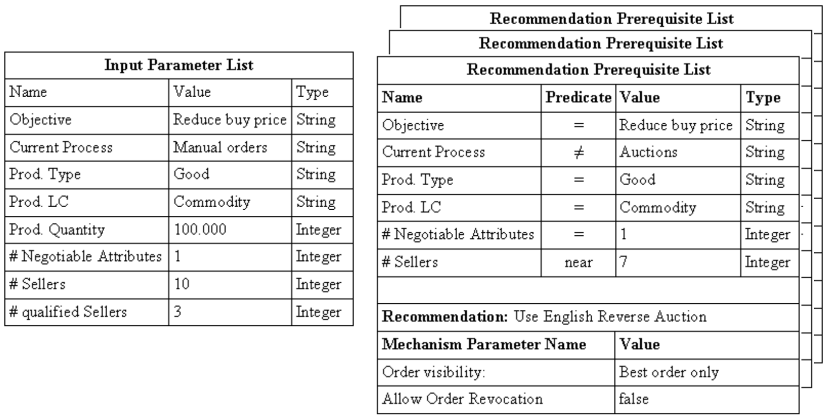
Fig. 4.10. Entity Relationship Model of the Knowledge Base

on proprietary storage formats that do not cope well with traditional DBMS. To the authors' knowledge only one (quite complex) approach exist that adapts the RETE algorithm to directly work on a database (Jin et al., 2005).

For MDKB a database for storage and retrieval of recommendations was more advantageous as it provides a convenient way to store verbal recommendations along with structured information and allows easy manipulation of the stored data. Thus an alternative approach for recommendation retrieval was developed, which stems from the research on recommender-systems. In this area, case-based reasoning is oftentimes used to compute similarities between a new case and existing (historic) cases (Chi and Kiang, 1991; Porter et al., 1993).

MDKB implements a case based reasoning algorithm that compares a new case (recommendation request) to cases (recommendations) already stored in MDKB. Like this the task of finding a suitable mechanism recommendation can be reduced to comparing parameter lists with each other and returning one list if the number of matches between the list elements exceeds a certain predefined threshold value.

The first list (c.f. Figure 4.11) consists of parameters a user enters into the system in order to describe the procurement situation he seeks advice for. The second list contains parameters from the same parameter domain as the first list but in this case the parameters are prerequisites that must be fulfilled for a recommendation to be valid. Figure 4.11 shows schematic examples of these lists. For the recommendation retrieval, the input parameter list is compared with each recommendation prerequisite list stored in the knowledge base. For each comparison cycle the similarity between all list items from both compared lists is computed on a per attribute basis. If an attribute is found in the input parameter list but not in the respective recommendation prerequisite list, the parameter is counted as *relaxation*, as it is not necessary for the current recommendation to be valid. If a parameter on the other hand is only found in the recommendation prerequisite list, it is counted as *restriction* as the parameter was not specified by the user but is required for the recommendation to be valid.



Input Parameter List

Name	Value	Type
Objective	Reduce buy price	String
Current Process	Manual orders	String
Prod. Type	Good	String
Prod. LC	Commodity	String
Prod. Quantity	100.000	Integer
# Negotiable Attributes	1	Integer
# Sellers	10	Integer
# qualified Sellers	3	Integer

Recommendation Prerequisite List

Recommendation Prerequisite List

Recommendation Prerequisite List

Name	Predicate	Value	Type
Objective	=	Reduce buy price	String
Current Process	≠	Auctions	String
Prod. Type	=	Good	String
Prod. LC	=	Commodity	String
# Negotiable Attributes	=	1	Integer
# Sellers	near	7	Integer

Recommendation: Use English Reverse Auction

Mechanism Parameter Name	Value
Order visibility:	Best order only
Allow Order Revocation	false

Fig. 4.11. Input Parameter List and Recommendation Prerequisite List

If a parameter is found in both lists, the similarity between both parameter values will be computed. It is counted as a match if the similarity exceeds an (adjustable) threshold level. Finally, after all comparisons, three measures are available indicating the matching quality of a recommendation:

- $\sharp\, restrictions$
- $\sharp\, relaxations$
- $matching\, quality := \frac{\sharp\, matching\, parameters}{\sharp\, parameters\, compared\, overall}$

A recommendation is returned to the user if (i) its matching quality exceeds a predefined threshold, (ii) the number of restrictions does not exceed a predefined threshold, and (iii) none of the *"restriction"* parameter was marked as *"knock-out"* criterion. For convenience, the results are sorted by matching quality in descending order. The number of relaxations and the number of restrictions are also displayed in the result list as further matching quality indicators.

Overall, MDKB is a system that is aimed at supporting market engineering with systematic decision support on which market mechanism to choose best in what type of market environment. This artifact can be used as an add-on during market development and was initially used in an industry project specifically focused on industrial sourcing (see Neumann et al., 2007; Block and Neumann, 2008). However, MDKB is not limited to procurement scenarios but can be used to store all types of market design recommendations in a parameterized and easily searchable format as shown in Figure 4.11 for arbitrary types of markets.

4.2.2 Market Repository

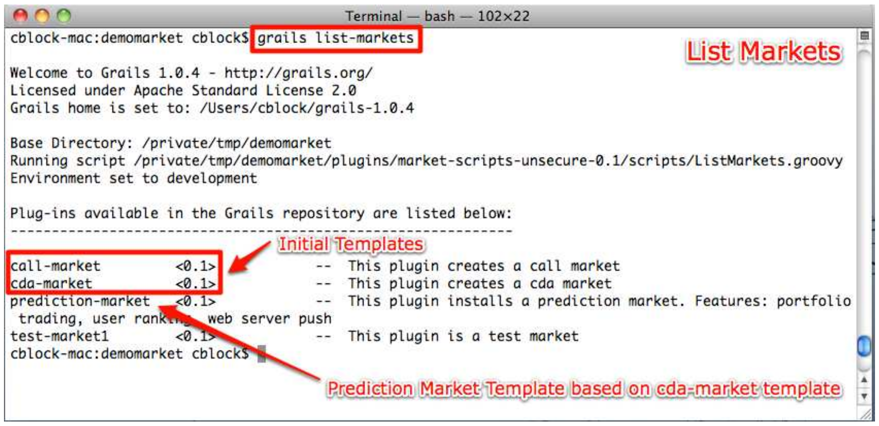
The market repository was created to support market developers in quickly instantiating new market instances from a set of previously developed and archived market platforms. Instead of developing new electronic markets from scratch in each new market engineering project, and instead of using generic but complex and hard to extend market runtime environments (see Section 2.2.2), small, simple, and highly specific market templates have been developed as foundations to build new electronic market projects upon. After initial instantiation of a specific market template, it can be incrementally customized to eventually meet all requirements of the envisioned new market platform. All market templates are immediately executable as web applications and exhibit default market functionality specific to the respective market template. For example, a continuous double auction market (CDA) template offers all matching, pricing, and settlement functionality as well as simple user, user-rights, and product management functions required to run a minimal but fully functional CDA market. The market provides a default web interface for human user interaction and a default web services interface for machine-to-machine communication which can be used e.g. by electronic trading agents. Like this, at the beginning of a new project, an initial version of the market platform can be set up, interactively tested and incrementally adjusted and refined with almost no installation and initialization overhead.

The market repository itself consists of a set of software artifacts. In particular, it comprises the following:

1. Specific **market templates** that implement one specific market mechanism (e.g. a call auction) each. Market templates are ready-to-deploy trading platforms with carefully chosen default conventions.
2. A **central repository** for market template storage and retrieval.
3. Several command line **scripts** (*list-markets*, *install-market*, *release-market*) that automate the work with the repository and the instantiation of market instances.

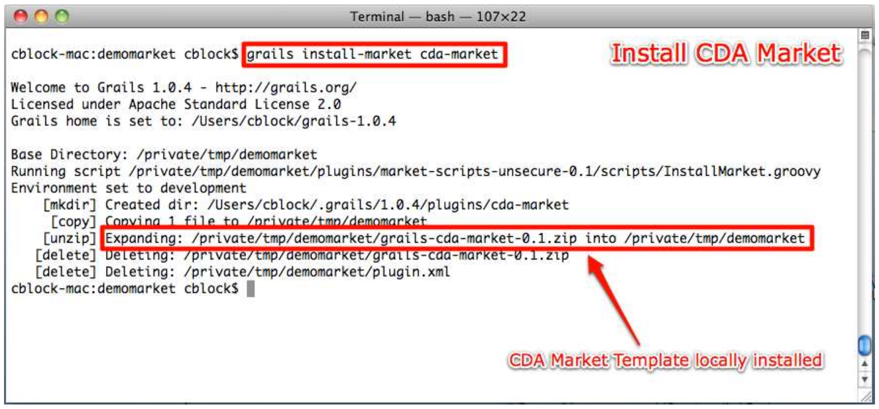
In Figure 4.16 the structure of the market repository as well as the interaction between market developer and market repository is shown. Initially (i.e. at the beginning of Step III of the agile market engineering process – see Section 4.1.1), the market developer lists all currently stored market templates available in the market repository as shown in Figure 4.12. Business owner, market expert, and market developer jointly choose the best fitting template for the current market development project, which is then instantiated via the command `install-market` as shown in Figure 4.13. Finally, the newly installed market instance (in the example here a CDA Market) is started as shown in Figure 4.14) using the `run-app` command script. This script automatically compiles the market's source code and then deploys the binary code to a locally running web application server. BO, MD, and ME can then immediately evaluate and test the running system (Figure 4.15 and all of its default functionality.

In subsequent development cycles, the market developer(s) then start to customize the instantiated market template in order to make it fit to the functional requirements of the respective project. After a new market instance has matured enough



```
cblock-mac:demomarket cblock$ grails list-markets

Welcome to Grails 1.0.4 - http://grails.org/
Licensed under Apache Standard License 2.0
Grails home is set to: /Users/cblock/grails-1.0.4

Base Directory: /private/tmp/demomarket
Running script /private/tmp/demomarket/plugins/market-scripts-unsecure-0.1/scripts/ListMarkets.groovy
Environment set to development

Plug-ins available in the Grails repository are listed below:
-------------------------------------------------------------
call-market         <0.1>         --  This plugin creates a call market
cda-market          <0.1>         --  This plugin creates a cda market
prediction-market   <0.1>         --  This plugin installs a prediction market. Features: portfolio
 trading, user ranking, web server push
test-market1        <0.1>         --  This plugin is a test market
cblock-mac:demomarket cblock$
```

Fig. 4.12. List available market templates in the Market Repository



```
cblock-mac:demomarket cblock$ grails install-market cda-market

Welcome to Grails 1.0.4 - http://grails.org/
Licensed under Apache Standard License 2.0
Grails home is set to: /Users/cblock/grails-1.0.4

Base Directory: /private/tmp/demomarket
Running script /private/tmp/demomarket/plugins/market-scripts-unsecure-0.1/scripts/InstallMarket.groovy
Environment set to development
    [mkdir] Created dir: /Users/cblock/.grails/1.0.4/plugins/cda-market
     [copy] Copying 1 file to /private/tmp/demomarket
    [unzip] Expanding: /private/tmp/demomarket/grails-cda-market-0.1.zip into /private/tmp/demomarket
   [delete] Deleting: /private/tmp/demomarket/grails-cda-market-0.1.zip
   [delete] Deleting: /private/tmp/demomarket/plugin.xml
cblock-mac:demomarket cblock$
```

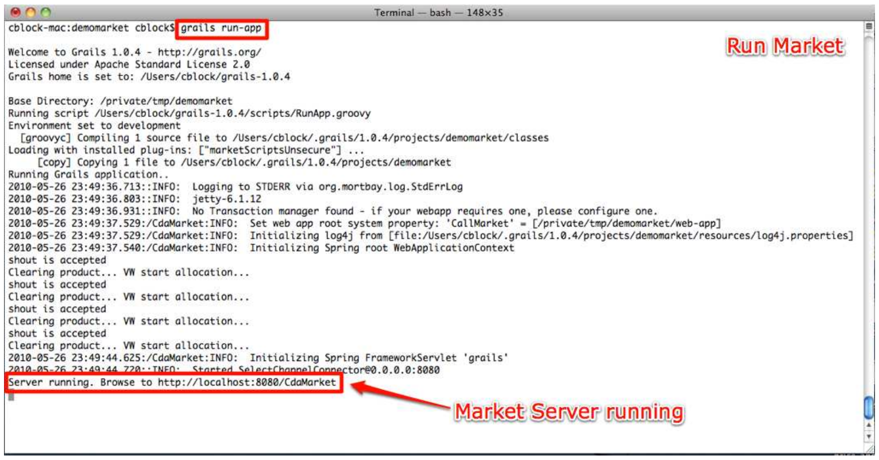
Fig. 4.13. Install a market template locally

(and if it provides significant new functionality as compared to the original market template), it can easily be contributed to the market repository as a new market template for other projects to build on.

Two initial market instances, a *call market* and a *continuous double auction market* were developed and contributed to the market repository as “base templates”. Both are described in more detail later in this chapter.

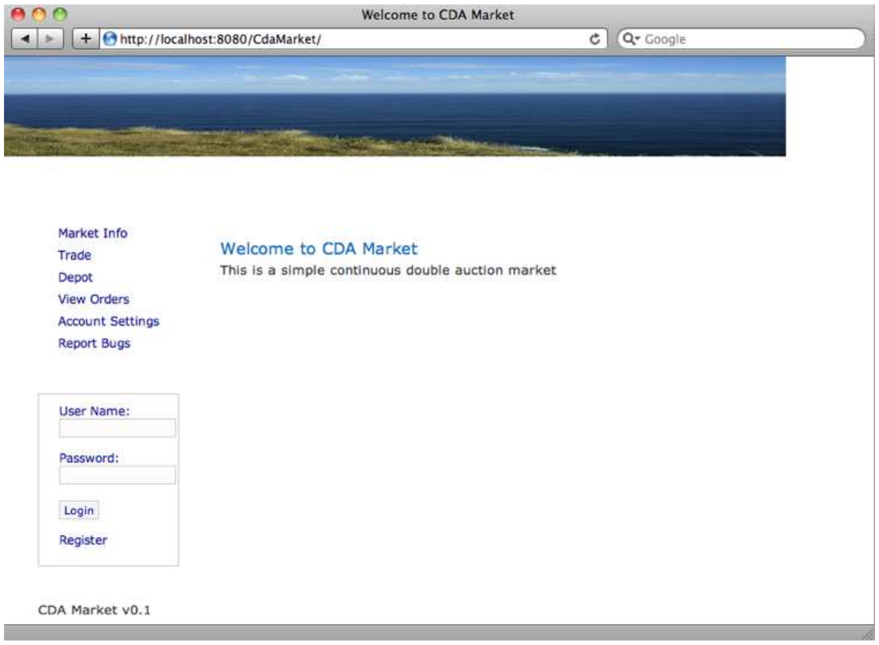
At the beginning of the development of the market repository and the aforementioned market templates, a number of different software development frameworks were evaluated. The initially considered frameworks included Grails[18], Ruby-on-Rails [19], Django[20], and TurboGears[21]. All of them follow the Model-View-Controller (MVC) design pattern as described in Gamma et al. (1995) and provide features like automated object relational mapping, extensibility via well defined plugin APIs, or

[18] `http://grails.org`
[19] `http://rubyonrails.org`
[20] `http://www.djangoproject.com/`
[21] `http://turbogears.org/`



```
cblock-mac:demomarket cblock$ grails run-app

Welcome to Grails 1.0.4 - http://grails.org/
Licensed under Apache Standard License 2.0
Grails home is set to: /Users/cblock/grails-1.0.4

Base Directory: /private/tmp/demomarket
Running script /Users/cblock/grails-1.0.4/scripts/RunApp.groovy
Environment set to development
  [groovyc] Compiling 1 source file to /Users/cblock/.grails/1.0.4/projects/demomarket/classes
Loading with installed plug-ins: ["marketScriptsUnsecure"] ...
     [copy] Copying 1 file to /Users/cblock/.grails/1.0.4/projects/demomarket
Running Grails application..
2010-05-26 23:49:36.713::INFO:  Logging to STDERR via org.mortbay.log.StdErrLog
2010-05-26 23:49:36.803::INFO:  jetty-6.1.12
2010-05-26 23:49:36.931::INFO:  No Transaction manager found - if your webapp requires one, please configure one.
2010-05-26 23:49:37.529:/CdaMarket:INFO:  Set web app root system property: 'CallMarket' = [/private/tmp/demomarket/web-app]
2010-05-26 23:49:37.529:/CdaMarket:INFO:  Initializing log4j from [file:/Users/cblock/.grails/1.0.4/projects/demomarket/resources/log4j.properties]
2010-05-26 23:49:37.540:/CdaMarket:INFO:  Initializing Spring root WebApplicationContext
shout is accepted
Clearing product... VW start allocation...
shout is accepted
Clearing product... VW start allocation...
shout is accepted
Clearing product... VW start allocation...
shout is accepted
Clearing product... VW start allocation...
2010-05-26 23:49:44.625:/CdaMarket:INFO:  Initializing Spring FrameworkServlet 'grails'
2010-05-26 23:49:44.720::INFO:  Started SelectChannelConnector@0.0.0.0:8080
Server running. Browse to http://localhost:8080/CdaMarket
```

Fig. 4.14. Run a newly installed market template locally



Welcome to CDA Market
This is a simple continuous double auction market

CDA Market v0.1

Fig. 4.15. Welcome Screen of a newly installed CDA Market instance

automatic generation (scaffolding) of simple web interfaces for standard use cases such as retrieving, adding, updating, or deleting of database objects. After a thorough comparison the Grails framework was chosen as software platform for the development of the market framework. Three main reasons guided this decision.

I. Grails offers the possibility to seamlessly combine Java and Groovy source code, which offers the conciseness and elegance of type-free languages on the one hand (Groovy) in combination with Java's execution speed, hardware system independence, and the rich variety of industry strength open source APIs (e.g. for authenti-

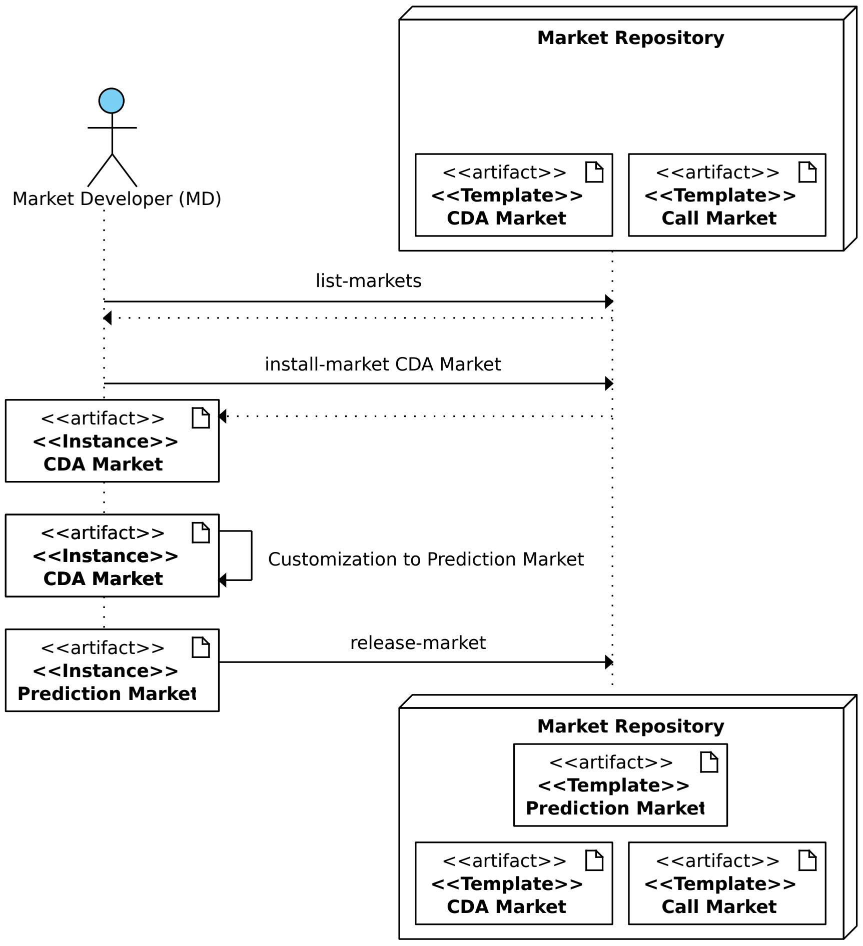


Fig. 4.16. Sample market engineering workflow using the market repository

cation/authorization[22], job scheduling[23], or database connectivity[24]. Electronic markets developed on top of this software infrastructure can thus be operated on any computer system a Java Runtime Environment (JRE) exists for and integrates well into the existing enterprise Java software infrastructure.

II. Staff at IISM had good experiences with Java as a programming language from previous market engineering projects (in particular from meet2trade project, see Section 2.3.3). A Java based framework thus offered the advantage of having well trained developers in the team who did not have to learn a new programming language.

III. Grails offers a powerful plugin concept and with more than 400 readily available plugins. Several of these can be used to support different steps in the agile market engineering process. In particular, unit, integration, and acceptance testing based on

[22] `http://static.springsource.org/spring-security/site/index.html`
[23] `http://www.opensymphony.com/quartz`
[24] `http://www.hibernate.org`

JUnit[25] and Selenium,[26], as well as code coverage and complexity metrics reporting based on Cobertura,[27] and CodeNarc[28] provide good support for market developers during step VI (Design, Implement & Test) of the development process. An adaptive logging framework based on log4j[29], as well as a special plugin for clickstream-logging[30] support market operators during step VIII in the agile market engineering process (Observe & Assess Market Activity).

For these reasons the Grails framework was finally chosen as the basis for the implementation of the market repository and its initial market templates.

The market repository itself extends the default Grails plugin management functionality. For its implementation, a source code management system for market templates was installed on a public Internet server. Then, particular Grails scripts for listing, installing, and publishing market templates were were developed. With this infrastructure at hand, the installation and execution of a new market instance can now be accomplished using three simple command line statements. Thus, a market developer can instantiate and start to adapt and refactor a market template at almost no effort at the beginning of the very first development phase. This allows a market developer to quickly deliver an initial, but fully functional and running version of the new market. The initial market templates consist of approx. 2000 Lines of Code (LOC) each (for a precise evaluation see Section 5.1), thus adaptation and customization of these instances is easier to accomplish than it was before on generic market platforms such as meet2trade. As most of the programmatic changes can applied to the code base while the market system is running, market developers can immediately see the changes in the corresponding web or web services interfaces.

Market Templates

Two initial market instances, a continuous double auction market (CDA), and a call market were developed and published in the market repository as base templates for future market development projects. Based on the requirements described in Section 3 both market templates were developed to provide

- a web-based interface for human market participants that uses reverse-ajax server push technology (Crane and McCarthy, 2008) for automatic delivery of market updates to the market participant's web browsers.
- a web services interface providing connectivity for external services such trading agents or decision support systems.
- adjustable functionality for user authentication and authorization.
- secure third-party authentication functionality by means of OpenID (Recordon and Reed, 2006) and LDAP (Howes et al., 1998) protocols.

[25] `http://www.junit.org/`
[26] `http://seleniumhq.org/`
[27] `http://cobertura.sourceforge.net/`
[28] `http://codenarc.sourceforge.net/`
[29] `http://logging.apache.org/log4j/index.html`
[30] `http://www.opensymphony.com/clickstream`

- provide integrated market information and statistics services for different levels of information granularity.
- have a small code base (approx. 2000 LOC) and are thus simple to maintain and to extend.
- possess an embedded build and deployment mechanism including an embedded application server (Tomcat[31]) and an embedded relational database (HSQLDB[32]), which allows the market platform to be executed on any computer system that has Java installed (see Figure 4.14).
- compliance with industry standards for enterprise software in order to be easily integrable into existing software application landscapes.
- rely on open source technologies and components where possible and are licensed under Apache 2.0 open source license.

The different prediction market projects as well as the TAC Energy project described in Chapter 5 are all based on these market templates.

Both, CDA and call market template use the same initial set of domain objects shown in Figure 4.17. These are used to persistently store all data required to run a CDA or a call market. A `Person` domain object represents a market participant and is used to store personal details like user name, password, email address, or activation state. Person objects can be assigned zero or more `Roles` (e.g. administrator, trader, ...). These in turn are used to grant particular permissions on the market platform to different groups of persons. For example, market participants with role “regular user” may trade products while users with role “Administrator” are allowed to manage the product catalogue. A `Product` domain object represents a good or service that can be traded on the market platform. It contains several properties such as product name, description, or activation state. To announce the willingness to buy or sell a product on the market, a `Person` creates a `Shout` domain object, which represents a buy or sell order.[33] A shout object contains the person's offer details for buying or selling a particular product. Specifically, order type (limit order or market order), limit price, order quantity, buy/sell indicator, and creation date of the order can be specified. A `DepotPosition` domain object describes the quantity a particular person owns of a particular product. More specifically, it describes (i) the relative change in product quantity a `Person` owns, and stores the market transaction that lead to this change. Moreover it stores the running balance of the product that the person owns at the point in time when the market transaction took place. For example, if a person owns 100 pieces of a product and buys 100 additional pieces of the same product a new DepotPosition object will be created with the property `change=100` and the property `absolute=200`. Like this, changes can be tracked and reconstructed over time. `CashPosition` objects are similar to DepotPositions but are used to track the relative change and the absolute balance of money a person possesses over time. Whenever a trade occurs (i.e. whenever a

[31] `http://tomcat.apache.org`
[32] `http://hsqldb.org`
[33] In most database systems the word “order” is a reserved word so that naming objects or database tables like this can cause runtime errors. The word *shout* was chosen following the naming conventions of Phelps (2007)

certain amount of a product is transferred from one person to another and the corresponding amount of money is transferred in reverse direction) `TransactionLog` and `Orderbook` objects are created that store transaction prices, trade quantities, and the new, updated orderbook states. Overall, this domain structure was designed to closely reflect Sirca's[34] data model for storing ThompsonReuters financial market data. Like this, statistics tools and services developed for financial market research can be easily applied to default market data from market platforms that build upon on of the templates.

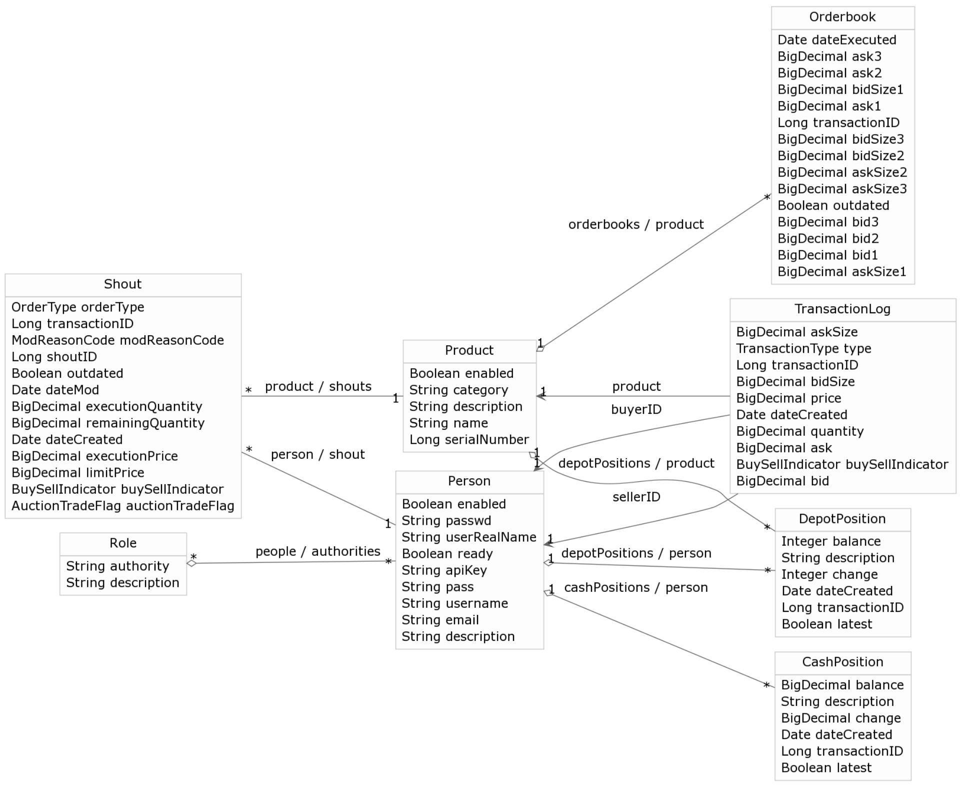


Fig. 4.17. Entity Relationship Diagram for CDA and Call Market Templates

Besides data persistence, the market templates also contain specific business logic that implements the market rules and provides all functionality required for market participants to trade on the market platform. All business logic code is collected in a set of services. In particular, the following services are provided by default:

- *Allocator Service* computes the product allocations
- *Auction Service* aggregates all sub services and exposes order entry and order update / delete functionality of the market platform to the web as well as to the web services interfaces

[34] `http://www.sirca.org.au/`

- *Authentication Service* authenticates users and controls their access to the functionality and data of the market platform
- *Billing Service* charges market participants a service fee for their market usage (e.g. for order submission or for trade execution)
- *Clearing Service* deletes outdated orders and clears the market by dispatching incoming orders to allocator, pricing, settlement, and billing services, and by triggering market information and market log service after a new transaction occurred.
- *Market Information Service* delivers market data to market participants
- *Market Log Service* performs logging of market transaction data for monitoring and analysis purposes
- *Pricing Service* determines transaction prices based on an allocation
- *Security Margin Service* (optionally) provides functionality that ensures that a market participant has enough money (or stocks respectively) available before a new order is accepted for execution
- *Settlement Service* settles trades by manipulating depot and cash accounts of the involved market participants

Detailed sequence diagrams that show the order processing workflow and the interaction of these services are provided in Appendix D for the default CDA market template and for the call market template respectively.

Conceptually, the approach of using small and very specific market templates to build new market platforms upon differs significantly from previous approaches in this field (see Section 2.3). GEM, AuctionBot and meet2trade relied on complex platforms that were adjusted by means of configuration to meet requirements for a particular market. This approach builds on the assumption that every single market project is *that* specific that its requirements can never be met by a generic default platform solution. Instead some specific customizing (and thus programming) always has to be accomplished. But the realization of these customizations needs to be as easy to accomplish as possible. Thus each single market consists of very concise and very specific code for one single type of market mechanism only. This reduces complexity and speeds up development. The CDA market template, for example, does not contain any logic required to run a call market and vice versa. Still they share (where possible) a set of conventions such as common names for domain classes and a common project directory structure that allows market developers to easily orient themselves in a new market projects due to these similarities. As a result small and simple but still fully functional market instances have been developed each consisting of approximately 2000 LOC only. A more detailed evaluation of the market templates is conducted in Section 5.1.

4.2.3 Market Simulation Framework

The market simulation framework provides business owner and market developer with a realistic and simple to use market simulation testbed that facilitates the evaluation of newly developed electronic market platforms in situations where the platform is technically developed but business owner and market expert do not feel confident enough to open it up for public access yet (see Section 4.1.2).

Similar to the concept of Collins et al. (2009) the market simulation framework was developed as a web-based system for defining and running market simulations, and for collecting results. In contrast to Collins et al. the market simulation framework is not limited to one specific market mechanism. It is a research tool, intended to support experimentation and evaluation of alternate market configurations in different environmental settings based on an agent-based computational simulation approach (Tesfatsion, 2002). With the market simulation framework, BO and MD are able to set up experiment series, each of which runs multiple market simulations with a fixed set of configuration parameters. Market and agent logs are gathered together for later analysis and stored centrally. Experiment series can be queued, and given enough hardware, multiple simulations can be run in parallel.

The market simulation framework consists of three core components:

Experiment Center: A web based interface for configuring and managing market simulation series.

Experiment Runner: A market and agent runtime environment, that executes pre-configured market simulations and can be remotely controlled from the experiment center.

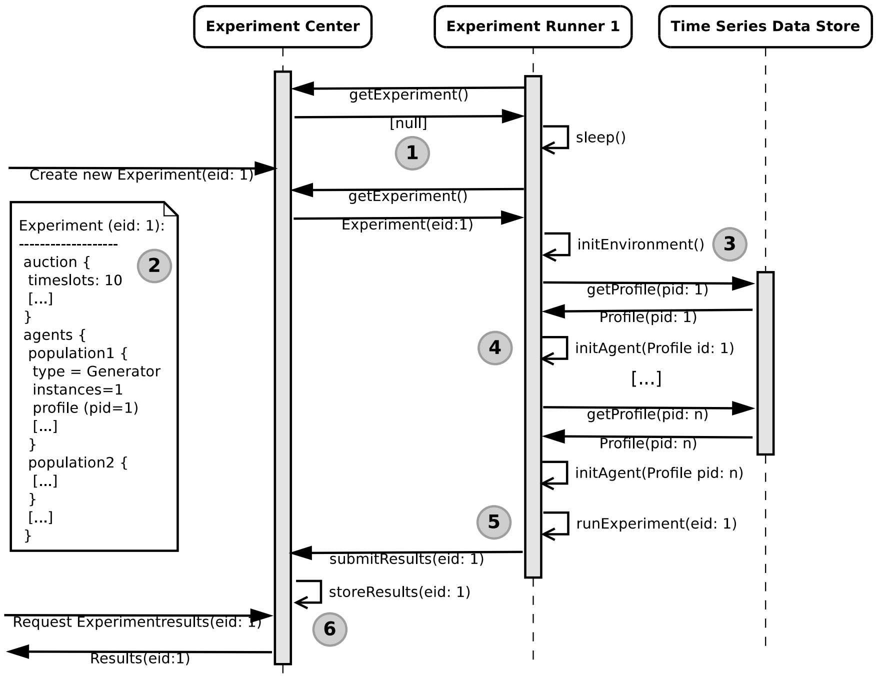
Time Series Data Store: A web based service for storage and retrieval of arbitrary historic time series data that can be used to initialize electronic agents with historic (market) data.

Figure 4.18 visualizes the general workflow and the interaction between the three market simulation environment components.

Initially, the experiment center is started and one or more experiment runner instances are deployed and started on different computers. Once an experiment runner instance is started, it stays in idle mode and periodically triggers the experiment center requesting new experiment instances to be executed (1). All experiment runners can be centrally controlled from the experiment center.

MD or BO can now create and execute market simulation series (experiments) using the experiment center's web interface (2). Each single simulation instance can be individually configured using a set of key-value pairs following the conventions of Phelps (2007) and Weidlich (2008).

Once a new simulation experiment (or a simulation experiment series) is added to the experiment center the experiment details (in particular the experiment id and the experiment configuration) are distributed to the next available experiment runner instance. Based on the given experiment configuration, the experiment runner then configures the market runtime environment initializing market and electronic trading agents as prescribed by the configuration (3). During initialization of the trading agent instances, some of them might need to be bootstrapped with historic time series data from the time series data store (4). This task of fetching the required timeseries data and providing it to the respective trading agent is completely automated and requires no manual user interaction.

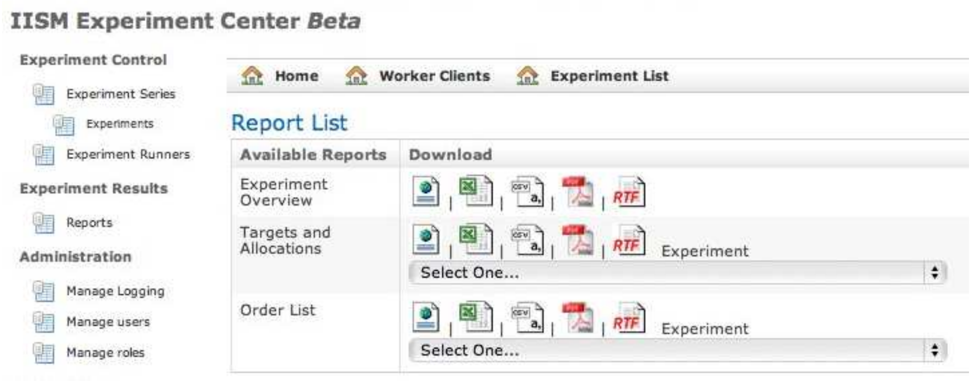


```
Experiment (eid: 1):
-------------------
 auction {
  timeslots: 10
  [...]
 }
 agents {
  population1 {
   type = Generator
   instances=1
   profile (pid=1)
   [...]
  }
  population2 {
   [...]
  }
  [...]
 }
```

Fig. 4.18. Market Simulation Framework components and their interaction

After the environment and all agent instances are properly initialized, the market is started and the agents begin trading (5). All activity in the simulation run is monitored by the experiment runner, which logs market as well as agent activity and – after the end of the simulation run – transfers all of the data back to the experiment center (6) where is is permanently stored for later analysis.

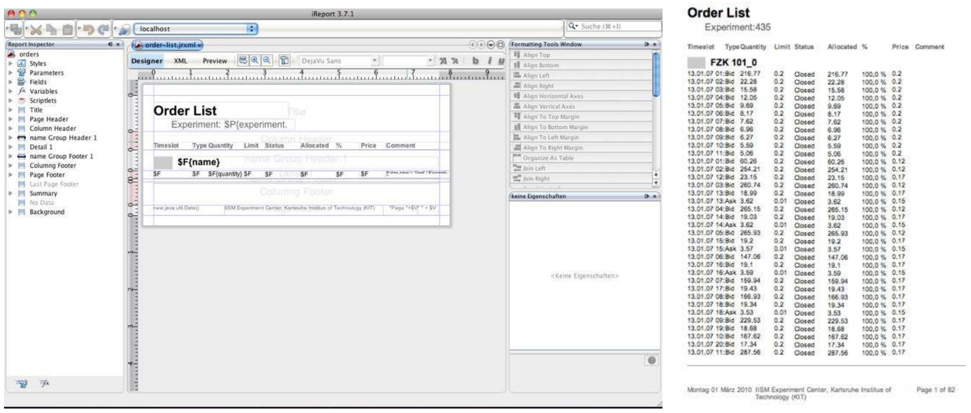
The experiment center has a data analysis and reporting center included that allows users to define and execute individual reports and to download the results in several different file formats as shown in Figure 4.19.



Available Reports	Download
Experiment Overview	
Targets and Allocations	Experiment Select One...
Order List	Experiment Select One...

Fig. 4.19. Report Management in the Experiment Center

Technically, the experiment center's reporting functionality builds on top of the open source reporting framework JasperReports[35]. This allows users to graphically design specific data analysis reports using JasperReport's graphical report designer. Reports are stored as specific xml files, which can be easily uploaded to the experiment center then become immediately available for data analysis. Once a new report is available online in the experiment center it can be executed against simulation data stored in the experiment center. Figure 4.20 shows how a custom report is defined in the JasperReport graphical report designer as well as some sample results after it was uploaded to the experiment center and executed against stored simulation data.



Order List

Experiment:435

Timeslot	Type	Quantity	Limit	Status	Allocated	%	Price	Comment
FZK 101_0								
13.01.07 01	Bid	216.77	0.2	Closed	216.77	100,0 %	0.2	
13.01.07 02	Bid	22.28	0.2	Closed	22.28	100,0 %	0.2	
13.01.07 03	Bid	15.58	0.2	Closed	15.58	100,0 %	0.2	
13.01.07 04	Bid	12.05	0.2	Closed	12.05	100,0 %	0.2	
13.01.07 05	Bid	9.69	0.2	Closed	9.69	100,0 %	0.2	
13.01.07 06	Bid	8.17	0.2	Closed	8.17	100,0 %	0.2	
13.01.07 07	Bid	7.62	0.2	Closed	7.62	100,0 %	0.2	
13.01.07 08	Bid	6.96	0.2	Closed	6.96	100,0 %	0.2	
13.01.07 09	Bid	6.27	0.2	Closed	6.27	100,0 %	0.2	
13.01.07 10	Bid	5.59	0.2	Closed	5.59	100,0 %	0.2	
13.01.07 11	Bid	5.06	0.2	Closed	5.06	100,0 %	0.2	
13.01.07 01	Bid	60.26	0.2	Closed	60.26	100,0 %	0.12	
13.01.07 02	Bid	254.21	0.2	Closed	254.21	100,0 %	0.12	
13.01.07 12	Bid	23.15	0.2	Closed	23.15	100,0 %	0.17	
13.01.07 03	Bid	260.74	0.2	Closed	260.74	100,0 %	0.12	
13.01.07 13	Bid	18.99	0.2	Closed	18.99	100,0 %	0.17	
13.01.07 13	Ask	3.62	0.01	Closed	3.62	100,0 %	0.15	
13.01.07 04	Bid	265.15	0.2	Closed	265.15	100,0 %	0.12	
13.01.07 14	Bid	19.03	0.2	Closed	19.03	100,0 %	0.17	
13.01.07 14	Ask	3.62	0.01	Closed	3.62	100,0 %	0.15	
13.01.07 05	Bid	265.93	0.2	Closed	265.93	100,0 %	0.12	
13.01.07 15	Bid	19.2	0.2	Closed	19.2	100,0 %	0.17	
13.01.07 15	Ask	3.57	0.01	Closed	3.57	100,0 %	0.15	
13.01.07 06	Bid	147.06	0.2	Closed	147.06	100,0 %	0.17	
13.01.07 16	Bid	19.1	0.2	Closed	19.1	100,0 %	0.17	
13.01.07 16	Ask	3.59	0.01	Closed	3.59	100,0 %	0.15	
13.01.07 07	Bid	159.94	0.2	Closed	159.94	100,0 %	0.17	
13.01.07 17	Bid	19.43	0.2	Closed	19.43	100,0 %	0.17	
13.01.07 08	Bid	166.93	0.2	Closed	166.93	100,0 %	0.17	
13.01.07 18	Bid	19.34	0.2	Closed	19.34	100,0 %	0.17	
13.01.07 18	Ask	3.53	0.01	Closed	3.53	100,0 %	0.15	
13.01.07 09	Bid	229.53	0.2	Closed	229.53	100,0 %	0.17	
13.01.07 19	Bid	18.68	0.2	Closed	18.68	100,0 %	0.17	
13.01.07 10	Bid	167.62	0.2	Closed	167.62	100,0 %	0.17	
13.01.07 20	Bid	17.34	0.2	Closed	17.34	100,0 %	0.17	
13.01.07 11	Bid	287.56	0.2	Closed	287.56	100,0 %	0.17	

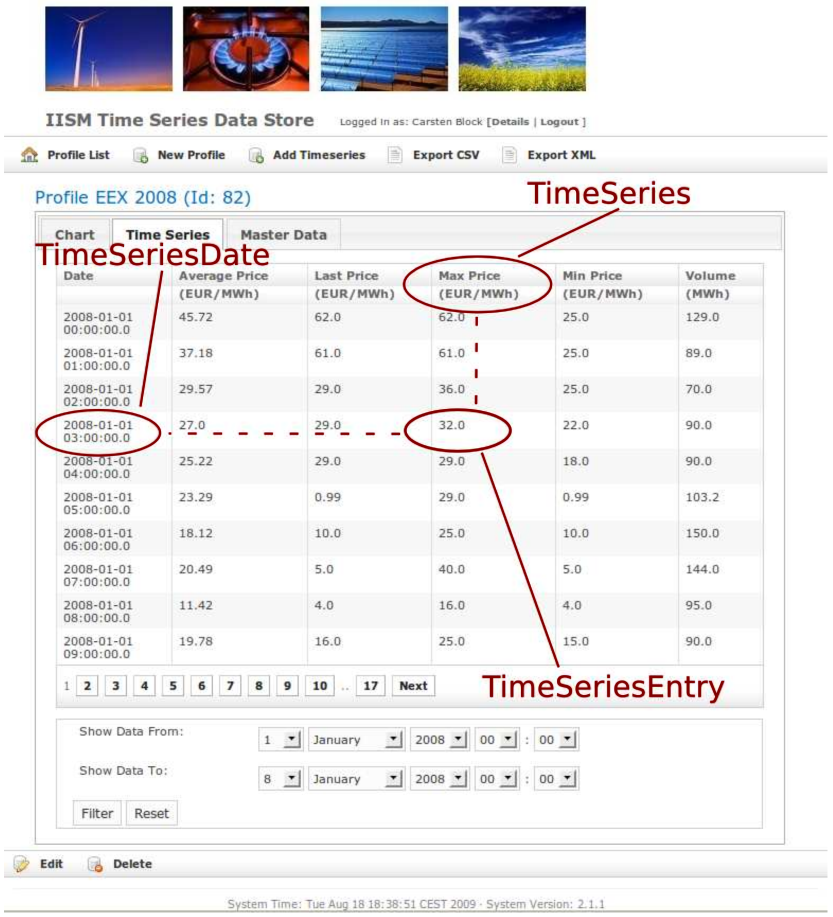
Montag 01 März 2010 IISM Experiment Center, Karlsruhe Institue of Technology (KIT) Page 1 of 82

Fig. 4.20. Graphical report creation and sample report results

Time Series Data Store A distinct feature of the simulation framework is its time series data store, which is designed to provide (i) flexible and fast storage for large amounts of historic time series data from different data sources and (ii) provides several different interfaces for data retrieval, each of them being restrictable through a flexible permission management system. Besides storage and retrieval for historic time series data, a plugin infrastructure for artificial time series generators is provided as well. As of writing of this thesis, a first plugin for the creation of artificial driving profiles for electric vehicles was realized and made publicly available (Dietz et al., 2010). Figure 4.21 shows a sample screenshot of one of the historic data profiles (in this case historic trading data) stored in the the time series data store.

A data `Profile` groups one ore more `TimeSeries`. It contains the master data for one or more interrelated time series, in particular profile name, description, data licensing conditions, and (if applicable) mail address, geographic coordinates as well as several other information like tags or contact details of the data provider. A `TimeSeries` contains information about the respective time series itself, most importantly the time series' name and its measurement unit. A profile also references one or more `TimeSeriesDate` objects. `TimeSeriesDates` store date and time information in a compact format (Unix Timestamp) as well as split up into separate date and time fields (i.e. year, month, day, hour, minute, second, millisecond). Like this, searching

[35] `http://jasperforge.org/projects/jasperreports`



IISM Time Series Data Store Logged in as: Carsten Block [Details | Logout]

Profile EEX 2008 (Id: 82)

Date	Average Price (EUR/MWh)	Last Price (EUR/MWh)	Max Price (EUR/MWh)	Min Price (EUR/MWh)	Volume (MWh)
2008-01-01 00:00:00.0	45.72	62.0	62.0	25.0	129.0
2008-01-01 01:00:00.0	37.18	61.0	61.0	25.0	89.0
2008-01-01 02:00:00.0	29.57	29.0	36.0	25.0	70.0
2008-01-01 03:00:00.0	27.0	29.0	32.0	22.0	90.0
2008-01-01 04:00:00.0	25.22	29.0	29.0	18.0	90.0
2008-01-01 05:00:00.0	23.29	0.99	29.0	0.99	103.2
2008-01-01 06:00:00.0	18.12	10.0	25.0	10.0	150.0
2008-01-01 07:00:00.0	20.49	5.0	40.0	5.0	144.0
2008-01-01 08:00:00.0	11.42	4.0	16.0	4.0	95.0
2008-01-01 09:00:00.0	19.78	16.0	25.0	15.0	90.0

1 2 3 4 5 6 7 8 9 10 .. 17 Next

Show Data From: 1 January 2008 00 : 00

Show Data To: 8 January 2008 00 : 00

Filter Reset

System Time: Tue Aug 18 18:38:51 CEST 2009 · System Version: 2.1.1

Fig. 4.21. Profile with several time series

for subsets of time series like e.g. *"all entries in year x and month y"* can be quickly executed on the underlying relational database.

A particular `TimeSeries` object and a particular `TimeSeriesDate` object reference a `TimeSeriesEntry` domain instance, which only stores a single value (a decimal number or a String). With this domain topology in place "vertical data queries" (e.g. select all time series entries from time series "Max Price") as well as "horizontal data queries" (select all time series entries for 2008-01-01 03:00:00.0) can be efficiently retrieved from the relational database. The time series data store is publicly available at `http://ibwmarkets.iw.uni-karlsruhe.de`.

4.3 Summary

In this chapter an agile market engineering process model is described that is designed to address the shortcoming of existing market engineering process models (see

Chapter 3). In particular, the agile market engineering process model relies on short, incremental market development cycles and frequent user feedback as main sources of input for market development. Like this, experiences and insights from the market development process itself as well as end user experiences from the running market platform can be incorporated into the incremental development and improvement process.

The agile market engineering process model is complemented by a set of supporting software artifacts. The *Market Design Knowledge Base* provides a technical infrastructure for storing and maintaining insights and experiences form market development projects over time and across project teams. The *Market Repository* in combination with a set of initial *Market Templates* helps market developers to quick-start the development of a new market platform by technically building their code base on small but fully functional market templates. Like this, agile market engineering projects can start the development based on a running and fully functional default market, that is subsequently customized to the needs and requirements of the specific project. The *Market Simulation Framework* provides a convenient and simple to use testbed for evaluating newly developed market platforms based on agent-based computational simulations.

Several market engineering projects have been conducted that relied on this process model and the aforementioned software artifacts. These are described in more detail in the following chapter in the form of case studies.

5

Evaluation

In this chapter, the agile market engineering process model and its supporting software artifacts are evaluated. Section 5.1 analyses the developed cda and call market templates with respect to their technical properties. Subsequently, in Section 5.2, 5.3. 5.4, 5.5 several short case studies are presented where the agile market engineering process model has been applied as project management methodology and where agile market engineering software artifacts were used to support the development process. The TAC Energy project described in Section 5.6 is another case study, which represents the most complex application scenario so far and is thus described in greater detail. Section 5.7 compares the developed process model and its artifacts with the original requirements described in Chapter 3 and concludes the chapter with a summary of the main findings.

5.1 Assessment of Default Market Templates

In its core, the agile market engineering process model builds on short, incremental market development cycles and frequent user feedback in order to develop and to continuously refine and improve the electronic market platform. Consequently, refactorings and adaptations of the platform's code base occur frequently. It is thus important that the software infrastructure an agile market engineering project builds upon facilitates and eases frequent changes and improvements. Empirically the three metrics *cyclomatic complexity*, *non commentary lines of code (LOC)*, and *number of methods per class* have been shown to be good proxies for estimating development and maintenance efforts of software projects (see Section 3.2 for details). In short, low code complexity few number of methods per class as few LOC per class (and overall) have been shown to be well correlated with good maintainability and adaptability of program code(Kafura and Reddy, 1987; Grady, 1994; Kan, 2002).

For all service classes (i.e. for the complete business logic) of the basic cda and call market templates these metrics have been calculated. The results are presented in Table 5.1.

With one exception, the cyclomatic complexity (McCabe, 1976) of all of the markets' business logic is well below the value of 14 as recommended by Kafura and Reddy

	CDA Market			Call Market		
Class name	max. CC	LOC	#Methods	max. CC	LOC	#Methods
`AllocatorService`	17	92	1	17	91	1
`AuctionService`	12	81	2	12	81	2
`AuthenticationService`	2	18	2	2	16	2
`BillingService`	1	27	3	1	27	3
`ClearingService`	6	71	2	6	67	2
`MarketInformationService`	4	157	7	4	148	7
`MarketLogService`	2	37	2	2	37	2
`PricingService`	1	12	1	1	12	1
`SecurityMarginService`	6	81	4	6	81	4
`SettlementService`	7	130	5	7	128	5
Total	**93**	**706**	**29**	**93**	**688**	**29**

Table 5.1. Software metrics for the business logic of CDA & Call market template

(1987); Grady (1994). Only for `AllocatorService` (the component that computes the allocated set of orders, see Section 4.2.2) the cyclomatic complexity exceeds the value of 14 by 3 points. Still, on average both market templates comply with recommendations from empirical literature. The maximum number of LOC per class is 157, though each single method is well below 100 LOC (see Appendic 5.1). The maximum number of methods per class (7 in `MarketInformationService`) is small too. Thus, also these metrics are well within common recommendations (see e.g. Kan, 2002). Appendix E contains the complete complexity analysis for both market templates. In total the call market template consists of 2174 LOC; the cda market template is composed of 1903 LOC. For comparison: The meet2trade's ARTE core platform (not including the adaptive trading clients as main user interfaces) consists of 13,811 methods, written in 847 classes, and divided into 92 code packages. The maximum measured cyclomatic complexity is 185 (class `org.efits.Trade.Markt.Allocator_CDA_III`), the maximum LOC per class is 3368 (class `org.efits.Trade.MetaMarkt.MetaMarkt`), which also contains the most methods per class (282).

In summary, the low cyclomatic complexity as well as the small number of methods and LOC in the cda and call market templates are good indicators for the manageability, adaptability, and comprehensibility of the templates' code bases.

These analytical findings were confirmed throughout a series of applied agile market engineering projects, which have been conducted over the last years. All of them built their code bases on one of the two aforementioned market templates and all of them used the agile market engineering process model for project management. The following sections describe these projects in more detail and show that the different project teams were able to quickly develop and operate new electronic market platforms using the agile market engineering process model and building their market platforms on top of one of the market templates analyzed in this section.

5.2 Use Case EM-Stoxx Market

EM-Stoxx was the first publicly accessible market platform that was built using the agile market engineering process model and its accompanying software framework. The development of EM-Stoxx was jointly lead by Stefan Luckner at University of Karlsruhe and Stephan Stathel at Research Center of Informatics (FZI). Project Kick-Off was April 07, 2008 two months ahead of the European Football Championship, which provided the framing for the project. The main objective was to evaluate the influence of electronic market maker agents on the forecasting quality of prediction markets (Stathel et al., 2008). Consequently, an electronic prediction market platform was created that allowed registered users to trade stocks of all soccer teams participating in European Soccer Cup 2008 via Internet. Additionally the market platform comprised electronic market maker agents that automatically generated buy and sell orders for each stock in the market following a pre-defined trading strategy.

Immediately after the project start, the team (the author of this thesis was involved as change manager, see Section 3.3.1) decided to choose the cda market template as the technical basis to build the new prediction market platform upon. This choice was natural because the new prediction market was expected to utilize a continuous double auction as its underlying market mechanism.

After instantiation and execution of the cda market, a first assessment of the default functionality provided by the market template was conducted and the following additional requirements were identified. (i) A user ranking functionality was required that was capable of producing a list of all registered market participants sorted according to their respective overall depot values. Furthermore, (ii) portfolio trading (Graefe and Weinhardt, 2008), (iii) automated market making functionalities realized by means of electronic trading agents, and (iv) the possibility to assign signed-up market participants to different, identically looking but completely independently running, *market segments*, were identified as additional key requirements.

At the end of the first project week, the portfolio trading and user ranking functionality was implemented and tested, which required an overall effort of about 10 FTE days (i.e. two market developers working full time for one week). Also, a simple but functional implementation of the required market maker agents was realized at an estimated effort of approximately 5 FTE days. Technically, four new services (i.e. classes containing the additional business logic) were created to extend the existing functionalities (see Section 4.2.2) as follows:

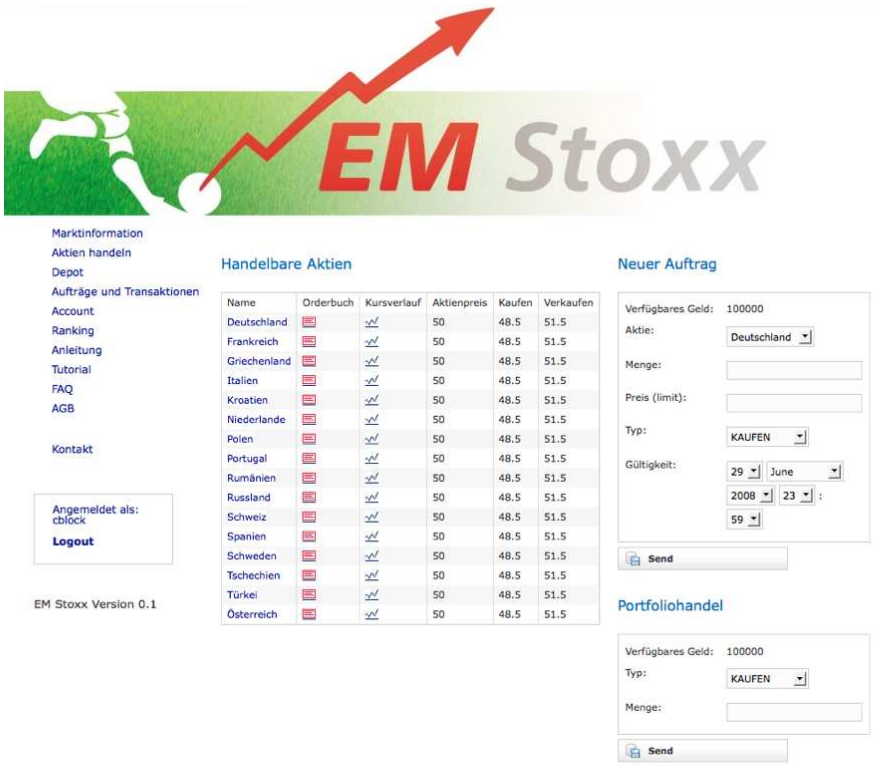
The *Ranking Service* comprised all business logic to compute aggregate portfolio values and to generate and update market participant ranking lists.

The *Portfolio Service* encapsulated the business logic required for portfolio trading.

The *Liquidity Agent Service* implemented the agent logic and the trading strategy of the liquidity provider, i.e. the automated creation of bids and asks with an adjustable spread size.

The *Portfolio Agent Service* encapsulated the portfolio agent's trading logic that was programmed to continuously observe the market, to calculated arbitrage potentials, and to realize these by issuing corresponding portfolio buy or sell orders.

Thus, with an overall effort of about 15 FTE days, the initial cda market template was extended and adjusted to become a basic but fully functional and running electronic prediction market platform. During the project weeks that followed the trading agent logic was refined and many small incremental improvements to the web interface were realized. Figure 5.1, for example, shows the new order entry interface of the EM-Stoxx market that provided current (and automatically updated) market prices right next to the order entry mask in order to help market participants to make their trading decisions based on the latest market data available. Parallel to the development of the market platform the production server environment (including database server, application server, firewall, etc.) was set up and tested, which took about 3 FTE days overall and was conducted by the market operator at FZI.



Marktinformation
Aktien handeln
Depot
Aufträge und Transaktionen
Account
Ranking
Anleitung
Tutorial
FAQ
AGB

Kontakt

Angemeldet als:
cblock
Logout

EM Stoxx Version 0.1

Handelbare Aktien

Name	Orderbuch	Kursverlauf	Aktienpreis	Kaufen	Verkaufen
Deutschland	▤	📈	50	48.5	51.5
Frankreich	▤	📈	50	48.5	51.5
Griechenland	▤	📈	50	48.5	51.5
Italien	▤	📈	50	48.5	51.5
Kroatien	▤	📈	50	48.5	51.5
Niederlande	▤	📈	50	48.5	51.5
Polen	▤	📈	50	48.5	51.5
Portugal	▤	📈	50	48.5	51.5
Rumänien	▤	📈	50	48.5	51.5
Russland	▤	📈	50	48.5	51.5
Schweiz	▤	📈	50	48.5	51.5
Spanien	▤	📈	50	48.5	51.5
Schweden	▤	📈	50	48.5	51.5
Tschechien	▤	📈	50	48.5	51.5
Türkei	▤	📈	50	48.5	51.5
Österreich	▤	📈	50	48.5	51.5

Neuer Auftrag

Verfügbares Geld:	100000
Aktie:	Deutschland
Menge:	
Preis (limit):	
Typ:	KAUFEN
Gültigkeit:	29 June 2008 23 : 59

Send

Portfoliohandel

Verfügbares Geld:	100000
Typ:	KAUFEN
Menge:	

Send

Fig. 5.1. Order Entry Screen of the EM-Stoxx Market

The existing market services (settlement service, clearing service, etc.) did not have to be adjusted. Still, in a later phase of the market development, a logical separation of the overall market into separate, identical, but separately operated *market segments* was introduced. This functionality was required to randomly assign new signed-up market participants to one of two market segments. The first group traded in a market segment where liquidity provider and portfolio agent were activated, the other group served as a benchmark and traded in a market segment where no electronic agents were used at all. Both markets ran simultaneously. From a market participant's perspective both market segments looked completely identical though trading activities in each segment were completely separated from the other and market participants did not know about the market segmentation. To accomplish this task, the original domain model of the cda market template had to be extended. In particular, the `Product` domain class was extended to contain a `role` property. Afterwards the web interface of the market platform was restricted to only display those products to market participants that had the same role assigned as the market participant itself. Afterwards, during market operation, two sets of identical products were created and assigned either of two market roles *ROLE_GROUP1* or *ROLE_GROUP2*. Market participants that signed up for the EM-Stoxx market were randomly assigned to one of the two groups. The trading agents were restricted to only provide bids and asks for products assigned to *ROLE_GROUP1*. Realized like this, the market segmentation requirement was easily implemented at an overall effort of about 3 FTE days.

Every week, a new version of the EM-Stoxx market platform was released to the production server environment and made available to a closed group of beta testers. Based on the feedback of this group, new requirements for subsequent development phases were elicited in the form of user stories and implemented later on. Technically the management of the user stories was accomplished using the open source Buildix system[1], which provides an integrated infrastructure for continuous integration, source code management, documentation wiki, and an electronic taskboard for requirements tracking (see Step VI: Design, Implement & Test in Section 4.1.2).

The market performance and in particular the prediction quality in the market segment with agents was compared to that without agents during an ex-post analysis of the market data after the end of the European Football Championship . Stathel et al. (2008) provide details and information on the research agenda of this field project and report research results of the the experiment study itself.

In summary, during the course of the EM-Stoxx market project, a prediction market trading platform was developed within about 8 weeks of time and about 70 FTE days of overall market development effort. The market was launched on June 7th, 2008 and ran stable throughout the course of the European Football Championship, which ended June 29th 2008. Apart from insights pertaining to the projects underlying research agenda, several new insights on practical (prediction) market engineering were gained. Most prominently, it was recognized that untrained prediction market participants sometimes had difficulties to correctly translate their estimates into corresponding buy or sell orders on the market. This important finding for future prediction market projects was stored in the Market Design Knowledge Base (see Section 4.2.1).

After the project was finished, the EM-Stoxx market platform was considered being mature enough to become a market template on its own and thus was published as new *prediction market template* in the market repository. With the new knowledge on prediction market design and with a new prediction market template at hand

[1] `http://buildix.thoughtworks.com`

a follow-up project, the Australian Knowledge Exchange (AKX), was started. This project is described in more detail in the following section.

5.3 Use Case Australian Knowledge Exchange (AKX)

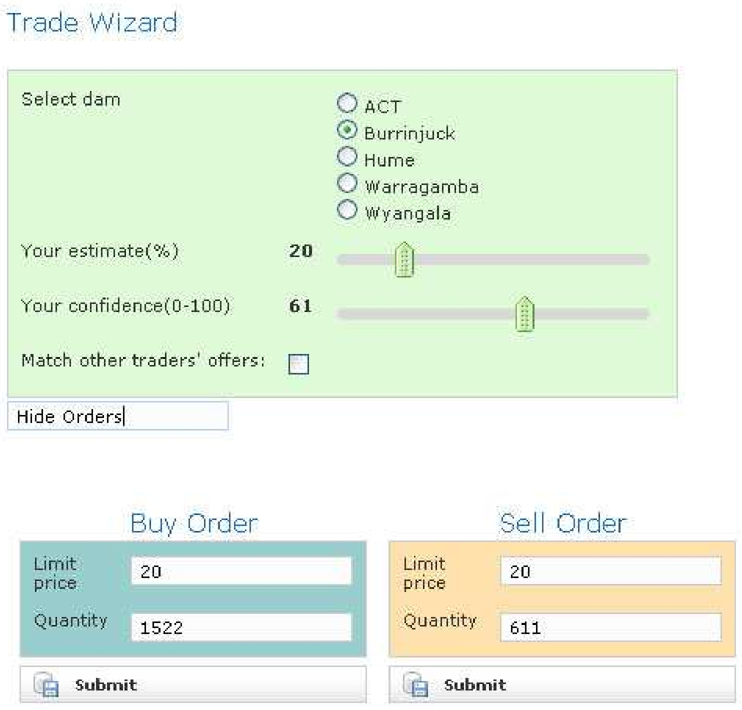
The Australian Knowledge Exchange was a joint project between Commonwealth Scientific and Industrial Research Organization (CSIRO), Australia and University of Karlsruhe, Germany. This research project started in September 2008 with the objective of exploring the forecasting potential of prediction markets in the area of water resource management. Thus, in this project, a prediction market platform had to be developed that allowed market participants to trade their expectations pertaining to fill levels of five important water reservoirs in Canberra and in southern New South Wales, Australia (Stathel et al., 2009). Florian Teschner from Institute of Information system and ,anagement at University of Karlsruhe had the technical lead for the realization of the market platform in this project.

At the beginning of the project, initial requirements for the AKX project were collected. An electronic market platform had to be developed, with water reservoirs as tradable stocks (products). The underlying real value of each stock was set to be equal to the (ex-ante unknown) fill level of the respective reservoir at a predefined date in the future (so called payout date) where all trading activity ended. AKX market participants had to be provided with an initial endowment of stocks and play-money (so called AKX Dollars), which they could use for trading. Participants of AKX were supposed to buy and sell stocks until the market's *payout date* was reached. At that date, all trading activity was suspended and all stocks of all participants were converted into money, each with the money equivalent of the respective reservoir's fill level. Assume, for example, a participant bought 10 “Hume Reservoir” stocks in the market for a price of $\$10AKX$, which equals to $\$100AKX$ for this transaction and this participant overall. If the reservoir's fill level at the payout date later on was measured to be, say, 50%, the participant's stock would have been converted in to AKX money at a price of $\$50AKX$ per stock, i.e. $10stocks \cdot \$50AKX = \$500AKX$ overall. Among the best market participants (in terms of overall money balance at the payout date) a price money of $50 AUD was distributed.

Besides the aforementioned basic prediction market functionality, the AKX project also required a market segmentation into a so called *expert market* and a *public market*. Access to the expert market was restricted to specially invited water experts from Australia who were supposed to trade their expectations in this market segment. In contrast, the public market was open for everybody to sign-up. The objective of this segmentation was to analyze ex-post if systematic differences between the expert and the public market segment occurred.

As in the EM-Stoxx market before both market segments had to provide completely identical functionality while market activity in both segments had to be completely separated. Different to EM-Stoxx market, no trading agent functionality was required.

Another key requirement, which stemmed from the insights of the previous EM-Stoxx market project, was the addition of some decision support functionality aimed at (optionally) guiding market participants in the process of converting their reservoir fill level expectations into corresponding buy or sell orders in the AKX market. Figure 5.2 shows the order entry wizard developed for the AKX market, which was offered to market participants as an additional alternative for entering buy or sell orders into the AKX market.


Trade Wizard

Select dam: ( ) ACT [✓ Burrinjuck] ( ) Hume ( ) Warragamba ( ) Wyangala

Your estimate(%) 20

Your confidence(0-100) 61

Match other traders' offers: ☐

Hide Orders

Buy Order

Limit price: 20

Quantity: 1522

Sell Order

Limit price: 20

Quantity: 611

Fig. 5.2. Order Entry Wizard of AKX

Using this order entry wizard, traders were enabled to simply choose the expected water dam levels and their own prediction confidence using simple slider bars. Based on this data the order entry wizard automatically generated appropriate buy or sell orders, which the market participant could check and submit to the market.

After the collection of the initial requirements and after an thorough inspection of the available market templates the project team chose the existing prediction market template as technical basis for the development of the AKX Market Platform. It was considered to be well suited for this project and provided most of the required functionality by default.

Technically, the market maker trading agent functionality, provided by the prediction market template by default had to be removed. The DSS functionality for supporting market participants during order entry only required changes to the web interface of the prediction market. Thus the conceptualization and realization of this functionality was accomplished with an effort of approximately 5 FTE days.

The development of the AKX market platform – including a complete customization of the web interface layout to an AKX branding – took an overall of about 25 FTE days. The market platform itself was operated stable and without incidents throughout the project runtime. Unfortunately only about 85 market participants signed up for the public market and only very few experts were attracted to trade their expectations in the expert market.

From a market engineering perspective, one of the key lessons learned from this project was the fact that the attractiveness of a prediction market depends in good parts on the frequent occurrence of events during market runtime that lead to "significant" changes in corresponding stock prices. During the AKX runtime, several heavy rainfalls occurred. But the corresponding changes in reservoir water levels were usually only in the magnitude of less than one percent and so did change the corresponding stocks on the AKX market, These were designed to be valued between 0 and $\$100AKX$ depending on the real percentage reservoir fill level. As a result stock prices usually changed only within the order of cents rather than dollars and the differences in the corresponding time series visualizations were small. This proved to be unattractive for traders who quickly seemed to loose interest in trading their expectations on the AKX market. Still, technically and mechanism wise the market platform ran stable and without incidents.

This insight was added to the market design knowledge base together with the recommendation for future prediction market projects to increase the degree of financial leverage of the traded stocks in situations similar to this one. In other words: In order to increase the prediction market's attractiveness, the fundamental stock value dynamics should be magnified e.g. in the AKX case by translating 1 per mille change in reservoir fill level (instead of 1 percent) into 1$ AKX change in stock value. Stathel et al. (2009) provide details on underlying research agenda of the project and its research findings.

5.4 Use Case EIX Market

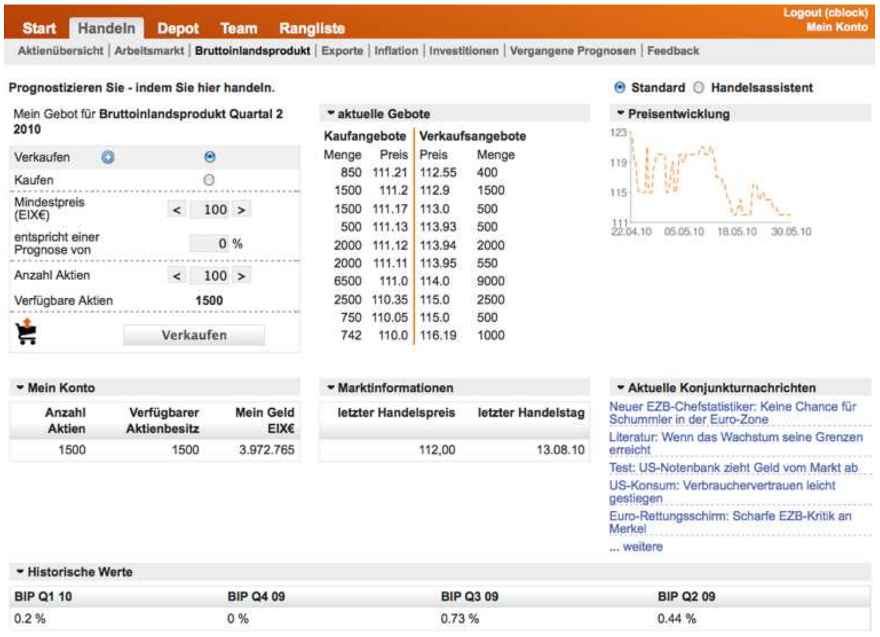
The Economic Indicator Exchange (EIX) is an ongoing cooperation project between Institute of Information Systems and Management (IISM), Research Center of Informatics (FZI), Institut der deutschen Wirtschaft Köln e.V. (IW), Handelsblatt GmbH & Co. KG, and Intuity Media Lab GmbH, which started in September 2009 and is still running at the time of writing of this thesis. The main objective of this project is to create, operate and evaluate an open electronic prediction market platform in order to forecast the development of several economic indicators (*gross domestic product*, *rate of inflation*, *unemployment*, *exports*, and *gross fixed capital formation*) for Germany.

Under the technical lead of Florian Teschner and Stephan Stathel, the development of the EIX market platform was started in September 2009 using the agile market engineering process model for project management. As in the AKX project before, the project development team decided to choose the prediction market template as base platform to build the EIX market upon.

Based on the experiences from the AKX market project, the key requirements for this project included (i) the development and implementation of more sophisticated activity rules for market participants to attract more trading activity on the market platform, (ii) a complete revision of the existing web interface (including examples and tutorials that describe the market rules) to further increase usability for market participants and to comply with corporate design guidelines of the project partners, (iii) a more sophisticated decision support assistant for order entry, and (iv) a detailed logging of market participant activity, in particular their usage of different types of decision support functionalities provided by the web interface of the EIX market for later analysis.

To realize the first requirement, the `RankingService` (see Section 5.2) was extended to calculate the number of transactions a market participant has concluded over time as well as the percentage change in his portfolio value. Under the activity rules of EIX market, only participants with more than five transactions per month and a relative increase in portfolio value qualify for participation in monthly prize money tombolas. The implementation and testing of these rules required an overall of 8 FTE days.

The complete revision of the prediction market's web interfaces, the creation of tutorials and trading examples (requirement II) was accomplished in close cooperation with Intuity, Handelsblatt, and IW. Here, the original decision to build the web interfaces of the underlying market templates – as far as possible – in pure HTML, facilitated the cooperation with the web and interface designers of the cooperation partners. Figure 5.3 shows the new order entry interface with (adjustable) information boxes that provide additional information (e.g. current orderbook, last trades, related news, etc.) providing the market participant with up to date market information during order entry.


Start
Handeln
Depot
Team
Rangliste
Logout (cblock)
Mein Konto
Aktienübersicht | Arbeitsmarkt | Bruttoinlandsprodukt | Exporte | Inflation | Investitionen | Vergangene Prognosen | Feedback
Standard
Handelsassistent
Verkaufen

Prognostizieren Sie - indem Sie hier handeln.

Mein Gebot für **Bruttoinlandsprodukt Quartal 2 2010**

Verkaufen	●
Kaufen	○
Mindestpreis (EIX€)	< 100 >
entspricht einer Prognose von	0 %
Anzahl Aktien	< 100 >
Verfügbare Aktien	1500

aktuelle Gebote

Kaufangebote		Verkaufsangebote	
Menge	Preis	Preis	Menge
850	111.21	112.55	400
1500	111.2	112.9	1500
1500	111.17	113.0	500
500	111.13	113.93	500
2000	111.12	113.94	2000
2000	111.11	113.95	550
6500	111.0	114.0	9000
2500	110.35	115.0	2500
750	110.05	115.0	500
742	110.0	116.19	1000

Preisentwicklung

Mein Konto

Anzahl Aktien	Verfügbarer Aktienbesitz	Mein Geld EIX€
1500	1500	3.972.765

Marktinformationen

letzter Handelspreis	letzter Handelstag
112,00	13.08.10

Aktuelle Konjunkturnachrichten

Neuer EZB-Chefstatistiker: Keine Chance für Schummler in der Euro-Zone
Literatur: Wenn das Wachstum seine Grenzen erreicht
Test: US-Notenbank zieht Geld vom Markt ab
US-Konsum: Verbrauchervertrauen leicht gestiegen
Euro-Rettungsschirm: Scharfe EZB-Kritik an Merkel
... weitere

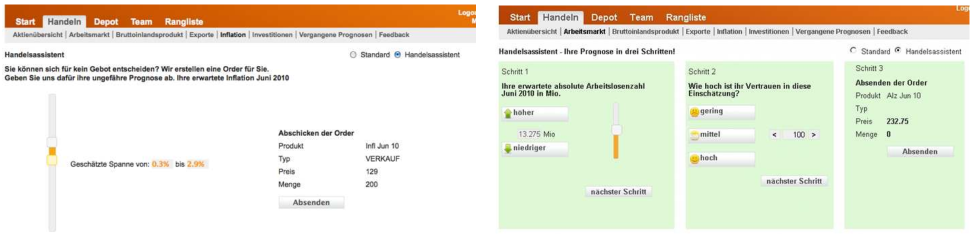
Historische Werte

BIP Q1 10	BIP Q4 09	BIP Q3 09	BIP Q2 09
0.2 %	0 %	0.73 %	0.44 %

Fig. 5.3. Order Entry Mask in EIX Market

The task of redesigning the interface and the creation of tutorials and help descriptions proved to be labour intensive though, requiring an overall of approximately 50 FTE days. The technical integration of the new web layout and the adapted market descriptions and tutorials, however, were easy and fast to accomplish requiring about 2 FTE days in effort.

Also, the implementation of order entry decision support assistants was easy to realize as it required changes only in the view layer of the market. Figure 5.4 shows two different order entry assistants developed for use in the EIX market.


Start
Handeln
Depot
Team
Rangliste
Aktienübersicht | Arbeitsmarkt | Bruttoinlandsprodukt | Exporte | Inflation | Investitionen | Vergangene Prognosen | Feedback
Standard
Handelsassistent
Absenden
höher
niedriger
gering
mittel
hoch
nächster Schritt

Handelsassistent

Sie können sich für kein Gebot entscheiden? Wir erstellen eine Order für Sie.
Geben Sie uns dafür ihre ungefähre Prognose ab. Ihre erwartete Inflation Juni 2010

Geschätzte Spanne von: 0.3% bis 2.9%

Abschicken der Order

Produkt	Infl Jun 10
Typ	VERKAUF
Preis	129
Menge	200

Handelsassistent - Ihre Prognose in drei Schritten!

Schritt 1

Ihre erwartete absolute Arbeitslosenzahl Juni 2010 in Mio.

13.275 Mio

Schritt 2

Wie hoch ist ihr Vertrauen in diese Einschätzung?

< 100 >

Schritt 3

Absenden der Order

Produkt	Alz Jun 10
Typ	
Preis	232.75
Menge	0

Fig. 5.4. Two different Order Entry Assistants for the EIX Market

The initial version of the EIX market platform was developed in the time between September and October 2009 and required an overall effort of approximately 120 FTE days (including the complete redesign of the web interface). Since the public launch of the platform on November 1st, 2009, about 950 traders have registered as market participants on the EIX prediction market platform. About 350 of these trade regularly.

Right now, wile the EIX market is running (Mai 2010), the development of the market platform still continues in short incremental cycles, which are used to implement improvements and ideas from the project team itself but also incorporate ideas based on feedback from market participants. Most prominently, user feedback elicited via the platform's online user feedback mechanism (see also Section 4.1.3 Step IX: Add & Revise Requirements) lead to an adaptation of the EIX platform's market rules. In a development phase in early February 2010, a short selling functionality for stocks was developed and subsequently released into the live EIX prediction market platform. Since then, participants have been able to express their predictions even in situations where they (temporarily) do not have sufficient stocks for sale in their depots. More details on this functionality are provided by Teschner and Storkenmaier (2010).

In summary, the initial EIX market development was successfully accomplished within a timespan of about two months and an overall effort of about 120 FTE days. The platform is running in stable operation since November 2009 without any major incidents and is currently serving a community of about 350 active prediction market participants on `http://www.eix-market.de`. The EIX market platform was developed based on the agile market engineering process model and is technically built upon the market repository's prediction market template. It is planned to replace the original prediction market template in the market repository with an improved version base on the current code base of the EIX market platform.

5.5 Use Case Microgrid Market

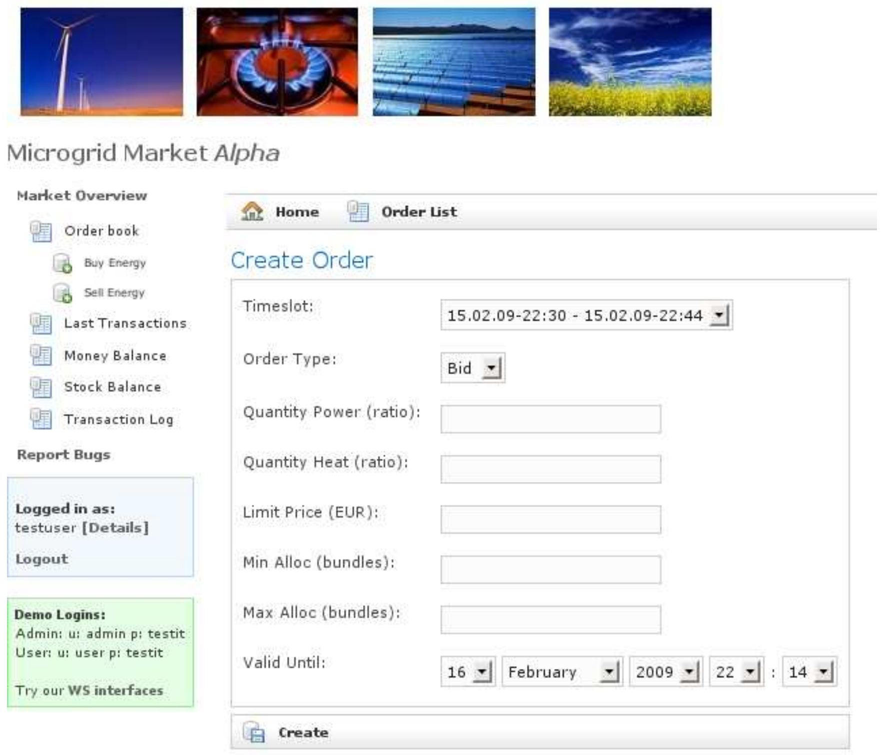
The microgrid market platform was developed from August 2008 onwards as a small project in preparation for the launch of the E-Energy MEREGIO project[2] in October 2008. Its main purpose was to develop a proof-of-concept electronic market platform and different types of specialized electronic trading agents for the dynamic scheduling of heat and power resources in microgrids equipped with a district heating infrastructure. The underlying economic concept and the market's mechanism design are described in more detail in Block (2007); Block et al. (2008).

The key requirements for this market platform were (i) the realization and timely demonstration of a combinatorial allocation and pricing mechanism capable of processing combinatorial heat and power orders in accordance with the market mechanism concept for microgrids developed in Block et al. (2008), and (ii) availability of a web services interface for electronic trading agents to automatically interact with the market during demonstrations.

After the project was started, the market development team (lead by the author of this thesis) decided to choose the call market template as initial code base to build the market platform upon. For the realization of the combinatorial allocation and pricing mechanisms the domain class model of the call market had to be extended

[2] http://www.meregio.de

first. In particular, the `Shout` domain class (see Section 4.2.2) was extended to contain additional properties for `exectuionQuantityHeat`, `executionQuantityPower`, `remainingQuantityHeat`, `remainingQuantityPower`, as well as `minAlloc` and `maxAlloc` properties. Additionally, the `DepotPosition` domain class was extended to contain `balancePower`, `balanceHeat`, and `changePower` and `changeHeat` properties. After the realization of these extensions, the market's domain model was capable of storing combinatorial orders e.g. from a trading agent representing a combined heat and power generator of the microgrid. Figure 5.5 shows the extended order entry mask of the microgrid market's web interface.



Create Order

Timeslot:	15.02.09-22:30 - 15.02.09-22:44
Order Type:	Bid
Quantity Power (ratio):	
Quantity Heat (ratio):	
Limit Price (EUR):	
Min Alloc (bundles):	
Max Alloc (bundles):	
Valid Until:	16 February 2009 22 : 14

Logged in as:
testuser [Details]
Logout

Demo Logins:
Admin: u: admin p: testit
User: u: user p: testit
Try our WS interfaces

Fig. 5.5. Order Entry Mask of the Microgridmarket

With the extended domain model at hand the `AllocatorService` and the `PricingService` had to be rewritten to implement the combinatorial allocation and pricing mechanism described in Block et al. (2008). The computation of the allocated set of orders requires solving a mixed integer problem. The adapted `AllocatorService` thus collects all new, unmatched orders available in the market at the time of market clearing and then builds a mixed integer optimization model out of them. This optimization model is handed over to the lp_solve[3] mixed integer problem solver, which computes the optimal order allocation. The result returned from the solver is translated into a set of allocated orders, which are then

[3] `http://lpsolve.sourceforge.net`

used by the adapted `PricingService` to compute the bundle prices for all orders in the allocation set. Finally, the adapted `SettelementService` updates the power and heat stocks of all participants whose orders have been allocated.

The technical realization of the microgridmarket platform required about 25 days FTE in overall effort (excluding the invention of the underlying combinatorial market mechanism).

Based on this market infrastructure, two different electronic trading agents, (i) a demand shifting agent representing private households (Deindl et al., 2008) and a CHP agent representing small-scale combined-heat-and-power plants were developed as part of two master thesis projects conducted by Matthias Deindl and Jan Huss.

As originally planned, the market platform and its trading agents were demonstrated live as proof-of-concept realization of a regional energy exchange market during the MEREGIO project kick-off meeting in October 2008. At the same time the code bases of the microgridmarket platform and the two trading agents was publicly released at `http://microgridmarket.sourceforge.net` and have been downloaded more than 400 times since then (as of May 2010). For the MEREGIO project itself, the microgridmarket trading platform has also been fitted into the market simulation framework (see Section 4.2.3) in order to run agent-based market simulations of the MEREGIO pilot region based on historic energy load and production data elicited from there simulating, for example, extreme events that cannot be studied in the field. This simulation concept is described in more detail in Hirsch et al. (2010).

Though principally attractive, the concept of the microgrid market platform and its trading agents as mechanism for efficient energy scheduling and energy distribution on an end consumer level is still in its early beginnings. One of the biggest obstacles that oppose a fast adoption of this concept is the fundamental lack of understanding of the market dynamics in situations where many different trading agents (fully or semi) autonomously compete against each other in acquiring or selling energy resources. Internationally, several large research projects have been conducted including EU Fenix (Jansen et al., 2009); ECN Powermatcher (Kok et al., 2005), EU CRISP (Schaeffer and Akkermans, 2006); US Olympic Peninsula (Hammerstrom et al., 2007), EU Microgrids (Hatziargyriou et al., 2006), or US CERTS (Lasseter et al., 2002). All of them rely on markets and price incentives in one form or the other in combination with intelligent automation technology for market interaction. And all of these projects showed in field trials that the principle concept of market-based energy management on the end consumer level works in a closed environment where all parameters can be controlled or at least significantly influenced by the respective project teams. But none of these project (and neither the microgrid market project) were able to demonstrate that their concepts are applicable in a competitive environment too where overall market control is limited and other market participants are likely to act or react in an unforeseeable manner. This insight lead to the idea of inventing an energy trading agent competition, which is described in more detail in the next section.

5.6 Use Case TAC Energy

By 2020, about 35% of the overall electricity demand in the European Union, EU, will be generated through distributed and intermittent "green energy" resources (Commission, 2009a). This presents a severe challenge to the existing energy infrastructure, which was designed to distribute power from a few large generating plants. It is thus very important to adopt a "smart" management approach that will facilitate effective integration of these resources into the existing energy generation and distribution infrastructure. Currently, much of the distributed generation capacity installed is virtually unmanaged, and cannot be supervised form centralized grid control and power dispatch systems.

With a small share of renewables in the overall generation capacity mix, this integration has not been a problem. But this strategy has almost reached its limits. Grid balancing capacities within the European Union for the Coordination of Transmission of Electricity (UCTE) high voltage grid are designed to cope with short-term variations in generator output of at most 3000 MW. Planned investments and projects underway in renewable capacity are expected to increase overall production volatility within the next five years to well beyond this level (Kox, 2009). Furthermore, the number of renewable producers and their wide-spread distribution will strongly increase. In Germany, for example, a roll-out of 100 000 distributed Combined Heat and Power (CHP) power plants has begun, with a planned overall installed capacity of 2 GW[4]. A purely centralized command-and-control approach to managing the grid and its generators has reached its limit, with the number of generators and the volatility of their output increasing even further over the next decade.

One approach to addressing this problem is to use the resource-allocation power of a market to find a near-optimal balance between producers and consumers of electrical power at lower levels of the grid hierarchy, and to make the participants in this market responsible for near-real-time balancing at the local level. So far, there is limited experience from pilot projects and field studies that could guide design and operation of such regional markets (Hatziargyriou et al., 2007; Blaabjerg et al., 2006). All these projects rely on "intelligent devices" and automation technology to facilitate or even automate energy management at both consumer and producer sites. But the automation solutions proposed within these projects are "unchallenged" in the sense that no other (competing) automation technology was deployed within these pilot projects, though competing technologies and solutions will be the default case in large-scale technology deployments on regional or national levels. The California energy market breakdown in 2000 (Joskow and Kahn, 2001; Borenstein et al., 2002) is an example of the problems that can occur if potential strategic, competitive or collusive behavior of market participants is not sufficiently accounted for in the design of such markets.

In the following the design and implementation of a competitive simulation environment is described that will address the need for solid research understanding of a market-based management structure for a local energy grid, which would mirror reality fairly closely. This simulation environment would challenge research teams to

[4] This is roughly the capacity equivalent of two nuclear power plants, see http://www.lichtblick.de/h/idee_302.php

create agents (Wooldridge and Jennings, 1995), or possibly agent-assisted decision support systems for human operators (Varga et al., 1994), that could operate effectively and profitably in direct competition with each other. Teams would also be challenged to exploit the structure of the market, and that structure would be adjusted periodically to defeat counterproductive strategic behaviors. The result would be a body of valuable research data that could guide energy policy, along with a much higher degree of confidence that such a mechanism could be safely introduced into operating energy systems.

The project is called "TAC Energy" because it is an example of a Trading Agent Competition[5] applied to energy markets. The main goals of the simulation are (i) to provide a competitive testbed for the development and validation of a market structure for managing electrical power distribution in a local grid, (ii) to spur research and development on intelligent agents and decision support systems that help automate decision processes in such markets, and (iii) to ease knowledge transfer between research and application by providing a testing environment that closely resembles reality. The entities competing in this market will broker electrical power in a local energy grid that contains a mix of intermittent energy sources, with residential, commercial, and industrial demands.

The next subsection gives background on the structure and organization of current electrical energy production and distribution systems, and reviews previous work both in competitive simulations and in other approaches to addressing the control and resource allocation problems in future energy grids. Then the essential decision problems are outlined that must be solved by a successful competing agent. Finally, the development of the TAC Energy market platform based on the agile software engineering process model is described before this case study description is concluded with a discussion of open research questions that can be addressed through a well-designed competitive simulation environment, many of which would be either impossible or highly risky to explore in the real world.

Background and Related Work for TAC Energy

In the following, portions of the existing electrical energy distribution systems that are relevant to the discussion in this paper are described in more detail. Subsequently some of the principal industry and government initiatives that are intended to resolve the problems of integrating new energy sources into the grid are discussed. Finally, related work in multi-agent modeling and competition testbeds is reviewed.

Energy infrastructure

Traditionally, the electrical energy infrastructure is organized in a strict hierarchy: A few centralized control facilities manage relatively few large power plants and schedule their production according to forecast energy demands, which are usually based on synthetic load profiles, i.e. average historic consumption time series for

[5] see www.tradingagents.org

different consumer groups. With these estimates at hand, generation capacities are dispatched. Traditionally, power is transmitted from power plants to consumers over a network of super-high, high, and medium voltage transmission grids into low voltage distribution grids that provide the connection to end customers.

With an increasing share of distributed, renewable generation capacities installed in the medium and low voltage grids, production volatility will increase and power flow inversions will occur, which the current technical grid infrastructure and control strategies are not designed for. In parallel, consumers are currently being equipped with smart metering technology and demand side management devices (DSM) that help them monitor and actively manage their loads. Consequently, their consumption flexibility will increase and load predictability via synthetic load profiles may become difficult. At the same time they will become at least somewhat responsive to time-varying energy prices.

In order to be able to manage this complex and highly dynamic system, the existing energy grid will have to be transformed into a smart grid with secure real-time ICT integration and communication among all of its components. Based on this information and communication infrastructure, an "Internet of Energy" will evolve (Block et al., 2008) that serves as a key enabler for new distributed and highly automated approaches to managing producers, consumer loads and the grid infrastructure.

Initiatives

The U.S. National Institute of Standards and Technology (NIST) recently published the first draft of a "Smart Grid Interoperability Standards Roadmap" (von Dollen, 2009). It defines a simplified domain model for a future smart grid with identified "Distribution", "Market," and "Customer" domains being in the core of the overall model. Furthermore a list of prioritized actions for the fast transformation of the current infrastructure into a smart grid is provided. Highest priority, according to NIST, are demand response and consumer energy efficiency measures. In particular they state that *"market information is currently not available to the customer domain. Without this information, customers cannot participate in the wholesale or retail markets. In order to include customers in the electricity marketplace, they need to understand when opportunities present themselves to bid into the marketplace and how much electricity is needed."*

In October 2009 the EU Commission announced the Strategic Energy Technology Plan (SET Plan) (Commission, 2009a) along with a draft technology roadmap (Commission, 2009b). One of the priority actions mentioned in this roadmap is the development of so called "smart cities" that efficiently and intelligently manage local energy production and consumption. In particular *"5-10 development and deployment programmes for smart grids in cities, in cooperation all relevant SET-Plan Initiatives, including priority access for local generation and renewable electricity, smart metering, storage, and demand response"* should be established within the next two years.

The BDI, a German industry group, has published a technology roadmap that describes the transition from the current energy infrastructure into a so called "Internet of Energy" (Block et al., 2008) on a timeline from 2009 to 2020. The document was written by a group of experts from various industries in cooperation with researchers from several different institutions. According to this roadmap, regional energy markets, virtual power plants based on micro CHP turbines, as well as DSM technologies will be mainstream by 2015, and one of the key challenges will be the development of *"applications and services for coordinating the energy grid on the business level."* In other words, the technical infrastructure will be in place but smart coordination and operation strategies are yet to be developed.

Multi-Agent Modeling

Electricity production and distribution systems are complex adaptive systems (Miller et al., 2007) that need to be managed in real time to balance the load of an electricity grid. Electricity markets are undergoing a transition from centrally regulated systems to decentralized markets (North et al., 2002). These transitions are very risky since there exists only limited experience in setting up decentralized energy markets and predicting their effect on the economy. Failures in designing such systems can cause major damage while deploying them in the real world. The California energy market (Borenstein et al., 2002), and the recent collapse of Enron, challenge the wisdom of deregulating the electricity industry, and have demonstrated that the success of competitive electricity markets crucially depends on market design, demand response, capacity reserves, financial risk management and reliability control along the electricity supply chain. Therefore, it is very important to thoroughly test system design proposals in a risk free simulated environment before deploying these ideas into the real world.

As energy systems move more toward open, competitive markets, the need for complex modeling systems becomes more obvious. Although traditional optimization and simulation tools will continue to provide many useful insights into market operations, they are typically limited in their ability to adequately reflect the diversity of agents participating in these markets, each with unique business strategies, risk preferences, and decision processes. Rather than assuming that the behavior of market participants is predictable, the TAC Energy project is aimed at using agent-based tools in a laboratory setting, which – on a technical level – extends the infrastructure of the Experiment Center described in Section 4.2.3. This framework will represent electricity markets and market participants using multiple agents, each with their own objectives and decision rules. Agents representing energy producers and consumers, the network operator and the regional wholesale market will be part of the simulation environment. The broker agents will be developed by competing research teams, thereby avoiding the cognitive blindness that can limit the value of results arising from non-competitive experimental work in such simulation environments.

Intelligent software agents (Wooldridge and Jennings, 1995; Jennings, 2000) offer many possibilities for automating, augmenting and coordinating business decision processes. These agents act on behalf of users, with some degree of independence or autonomy, employing some representation of the user's goals or desires. The focus in TAC Energy is on enhancing the adaptive learning component of such agents; the corresponding research is thus focused on trading agents that learn to operate effectively in competitive economic environments.

Agent-based modeling and simulation has emerged over the last few years as a dominant tool of the energy sector. For instance, the Electricity Market Complex Adaptive Systems Model (EMCAS) electric power simulation is an agent simulation that represents the behavior of an electric power system and the producers and consumers that work within it (North et al., 2002). In Sueyoshi and Tadiparthi (2008), the authors have developed MAIS, an agent-based decision support system for analyzing and understanding dynamic price changes for the U.S. wholesale electricity market before and during the California energy crisis.

Further, energy storage is one of the key underpinnings of the vision of the Smart Grid. In Vytelingum et al. (2010), the authors have developed a framework to analyze agent-based micro-storage management for the smart grid. Specifically, they designed a storage strategy (with an adaptive mechanism based on predicted market prices) for consumers and empirically demonstrated that the average storage profile converges towards a Nash equilibrium. Weidlich and Veit (2008) survey agent-based market models for wholesale electricity markets, and Zhou et al. (2007) review agent-based simulation tools and their application to the study of energy markets.

The field of Agent-based Computational Economics (ACE) (Tesfatsion, 2002) is the computational study of economic processes modeled as dynamic systems of interacting agents. Here "agent" refers broadly to a bundle of data and behaviors that represents an entity in a computationally constructed world. ACE models can support a variety of research agendas, such as understanding and evaluating market designs (McMillan, 2003; Marks, 2006), evaluating the interactions of automated markets and trading agents (MacKie-Mason and Wellman, 2006), creating rich economic decision environments for human-subject experiments (Duffy, 2006), and advising policy makers on the expected behaviors of markets or market interventions (Sun and Tesfatsion, 2007; Veit et al., 2009). A number of studies have used ACE methods to study electrical power markets, for example Nicolaisen et al. (2001); Conzelmann et al. (2004); Sun and Tesfatsion (2007).

The simulation approach used within TAC Energy extends ACE in the direction of developing strategies and decision procedures for competing agents in modeled market environments. Therefore, tools and methods of ACE, and in particular the infrastructure of the Market Simulation Framework developed as part of this thesis, are used to construct a rich simulated market environment in which one of the agent types faces competition from other agents of the same type. Researchers will be invited to implement their own agents to operate in that role, and pit them against each other in the simulated market. This provides a much more rigorous test of the market design, and produces deep knowledge of strategy options and decision procedures for these agents, such as the empirical game theory work of Jordan et al. (2007) or the economic-regime work of Ketter et al. (2009).

Multi-Agent Competitions

Along with the vision and development of the "Internet of Energy", a lack of intelligent, distributed energy coordination strategies is probably one of the key problems in the future of electricity production and distribution. In this context, agents are a promising tool to help solve these issues. This "TAC Energy" competition will provide a realistic, low-risk platform to effectively test and evaluate agent strategies and decision procedures. Over the last decade competitions are becoming increasingly prevalent in the research world. The current Trading Agent Competitions use a multi-year competition format to study trading in simultaneous online markets (TAC Classic) (Wellman et al., 2007), operation of a three-tier supply chain (TAC SCM) (Collins et al., 2005), trading of search keywords for advertising purposes (TAC AA) (Jordan et al., 2009), and the operation of online exchanges (TAC CAT) (Niu et al., 2010). The TAC CAT market design game explicitly seeks to encourage research in adaptive and automated mechanism design, a topic very closely related to our alternative market structures. CAT entrants compete against each other in attracting buyers and sellers and making profits. This is achieved by having effective matching rules and setting appropriate fees that are a good trade-off between making profit and attracting traders. The TAC CAT competition comes in spirit the closet to our TAC Energy competition. All these competitions are examples of crowd-sourcing, i.e. ask multiple, competing participants to innovate solutions to a problem (Howe, 2006).

Each research methodology has strengths and limitations. A well designed competition has many benefits (Stone, 2003). Much has been learned through designing, developing, and researching TAC SCM over the last eight years. This experience will guide the design of TAC Energy. For instance, one of the awkward features of TAC SCM is the design of the start and end of the scenario; agents start with no inventory at the beginning, and inventory has no residual value at the end of the scenario. TAC Energy will not have these problems, since the broker agents need to balance the energy grid in real time with extremely limited opportunity to store energy. Another shortcoming of TAC SCM is that the simulation is completely driven using predetermined statistical distributions. These distributions and parameters were carefully designed to balance the game among the three main agent types (supplier, manufacturer, and customer), but nevertheless they have been exploited in unrealistic ways by some research teams, and have generated much discussion among participants about the generalizability of results to real world scenarios. For the TAC Energy simulations a large body of anonymize real world energy consumption and production data should be used, which was collected from the campus energy management system at KIT, Karlsruhe, a small municipal utility in the middle of Germany, and (hopefully) also from a pilot region of the MEREGIO project covering 900 households in Germany (Hirsch et al., 2010). The data stored in and provided by the time series data store, which is part of the market simulation framework developed as part of this thesis. In case empirical data will not sufficiently be available, artificial time series generator plugins of the time series data store such as, for example, the artificial driving profile generator developed by Dietz et al. (2010)), will be used to generate artificial data series that still model reality as closely as possible.

TAC Energy has many more potential benefits as a research platform, such as the natural human desire to win. This will be a strong motivation to create a competitive agent by solving the challenges of the domain. Another advantage is that competitions force researchers to build complete, working systems by a specified date. Competitions such as TAC Energy can be strong research drivers for work in specific domains. For example, over the last seven years the annual TAC SCM tournament has pitted autonomous manufacturing agents against each other to determine which is the best performing agent. Each year, results from the previous competition have been published, such as Kiekintveld et al. (2009); Benisch et al. (2004); Ketter et al. (2009), which rapidly raises the bar for research. Since all teams have to solve the same problems in the same domain, this becomes a strong driver for research, and different teams learn from each other. Teams are encouraged to contribute their agents to a shared repository, allowing researchers to run controlled experiments that demonstrate the effect of individual agent designs, such as described by Jordan et al. (2007); Sodomka et al. (2007).

Market liberalization changes the landscape for corporate managers and public policy makers, who face difficulty in both predicting and understanding price changes in electricity markets. Price changes occur due to many uncontrollable factors such as changes in weather conditions, demographic changes, and different trading strategies among traders. Therefore, tools are needed to identify and predict market conditions in a dynamic fashion, such as the economic regimes method described in Ketter et al. (2009).

Competition Scenario

The competition is focused on the role of a broker acting as an "aggregator" of energy supply and demand, represented by a trading agent. According to von Dollen (2009) *"aggregators combine smaller participants (as providers or customers or curtailment) to enable distributed resources to play in the larger markets."*. In reality brokers could be energy retailers, municipal utilities, or cooperatives; in some cases large utilities could also take on the role of brokers.

Within the competition a broker sells tariff contracts to end customers (e.g. households, small and medium enterprises, owners of electric vehicles), which are attracted or deterred by the respective tariff conditions. Tariff conditions may include flat prices, time of use prices, peak prices, load caps for certain times of the day, contract runtime, etc. In addition to "classical" tariff contracts for energy consumption, a broker can also sell "energy production" tariffs to end customers. Under such a tariff agreement, a customer may be paid for operation of a decentralized energy generator such as a Combined Heat and Power (CHP) plant that feeds power into the grid.

Another type of special customers are Plug-in Electric Vehicles (PEVs). These customers receive special tariff contracts that have separate, time dependent prices for charging the vehicle (consuming energy) and for feeding energy back into the grid (effectively producing energy). As compared to households, PEV customers are comparably large energy consumers during their charge cycle but might decide to discharge some of their stored energy at their own discretion if the power generation prices are sufficiently attractive. Brokers may limit the maximum charge rate for PEVs (e.g. at most 16A), effectively throttling the speed of recharge, and they might encourage PEVs to feed energy back into the grid by setting the generation price appropriately, but they cannot directly enforce charging or discharging. This will allow PEV owners (and owners of other types of energy storage capacity) to engage in arbitrage by consuming power when prices are low and producing when prices are higher, thereby offsetting a portion of their capital costs.

On the tactical level (planning horizon: 1 week – 3 months) brokers have to manage their portfolios of consumer, producer and PEV contracts. On the operational level (planning horizon: 1 day – 1 month) brokers have to balance the fluctuating energy demands of their customers against the actual output of their contracted energy production capacity. Differences can be compensated through balancing power purchased or sold at an energy exchange.

The competition is designed to model most of these challenges, primarily from an economic rather than from a technical point of view. Real-world data is used where it is available, while keeping computational and technical complexity manageable.

In particular the following *assumptions* are made:

1. Within the simulated region, grid constraints (line capacity limitations) are assumed to be non existent, i.e. power flows within the region are unconstrained. Local distribution grids are typically overdimensioned with respect to their line capacities, thus this assumption is not a strong restriction but may have to be rethought in future once much more distributed generators and storage facilities are installed.

2. The point of common coupling (PCC) between the simulated distribution grid and the higher level transmission grid has a maximum capacity for power inflow and outflow. A specialized agent that serves as a "liquidity provider" on the regional energy spot market, and is able to arbitrage with the national energy spot market, has to obey these technical limits.

3. Power factor effects, i.e. phase shifts between voltage and current, are not taken into account. Modeling these effects would possibly influence the brokers' decision making on which consumers and producers to add to their portfolios but is out of scope at this time.

4. Power distribution and transformation losses are ignored. In Germany these losses are estimated at 6% (StatBA, 2008); for North America they are estimated at 5,5% (Program, 2006). These losses can be considered as being more or less constant within a distribution grid and identical for all grid participants. Thus the validity of the simulation results is not affected.

5. Two kinds of producers (energy production facilities) are distinguished. One kind (photovoltaic arrays, wind turbines) produce power when active, and are under control of their respective owners. The second kind (PEV batteries, some CHP units) is called "controllable" and may be switched on or off, or have its output adjusted remotely within its capacity range.

6. Technical load balancing (i.e. the real time operations of the local distribution grid) is accomplished outside the action domain of the competition participants using a combination of controllable generators and spinning reserves.

7. The simulation will model time as a series of discrete "timeslots" rather than as continuous time. This models the trading intervals in the wholesale market, and enables the simulation to model a period of days rather than minutes or hours.

8. The temporal distribution of energy consumption and generation *within* a timeslot is not taken into account. This means for example that balancing power de-

mand for a timeslot is calculated as the difference of the sum of generation and the sum of consumption for that timeslot and not as the instantaneous difference between the two timeseries.

9. Some portion of the load, including the charging and discharging of Plug-in Electric Vehicles, could be controlled by voluntary or automated means, using prospective or real-time price signals.

In order to expose the broker agents to tactical and operational decision making, the competition scenario proceeds through a series of alternating *contracting* and *execution* phases as depicted in Figure 5.6. The number of such phases will be indeterminate, to prevent strategic behaviors that exploit boundary effects. Both phases are described in more detail in the subsequent paragraphs.

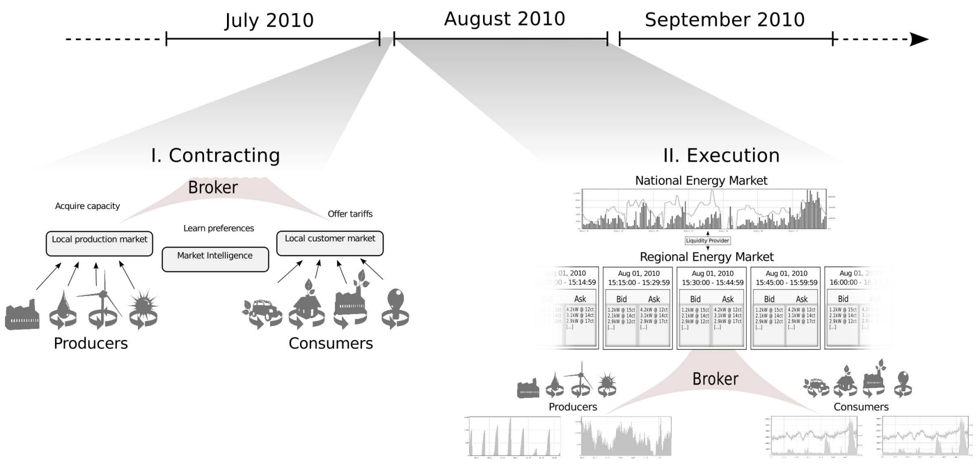


Fig. 5.6. Simulation timeline

To enhance the realism of the competition scenario, the simulation will be driven with real historical data on generation, consumption, and weather information, along with a model of preferences of potential customers derived from customer surveys and pilot projects. One potential source of such data series might be the German MeRegio project, a smart grid project that is implementing a combination of advanced grid control systems and innovative real-time pricing tariffs (Hirsch et al., 2010). The area around Freiamt currently serves as a pilot region for MeRegio; it contains a range of different distributed renewable energy generation facilities in combination with households and small and medium enterprises (SMEs) that are equipped with demand side management devices allowing them to flexibly react to price signals from the distribution grid.

With historic consumption and generation data collected from a region like this, the simulation environment exposes the broker agents to the challenge of managing virtual consumers and producers, which exhibit realistic energy consumption and generation patterns based on the history data.

Contracting phase

On the simulation timeline, a contracting phase represents a short period of time (perhaps 60-120 seconds). During this phase, broker agents try to acquire energy generation capacity from local producers and sell energy tariffs to local consumers.

Brokers can buy and sell energy through two different mechanisms. For most customers, such as households, small businesses, and small energy producers, brokers may offer tariffs that specify pricing and other terms. For large producers or consumers (for example, a large industrial facility or a greenhouse complex with many CHP units), brokers may negotiate individual contracts. During a contracting phase, brokers may simultaneously negotiate over individual contracts and offer tariffs as depicted in Figure 5.7.

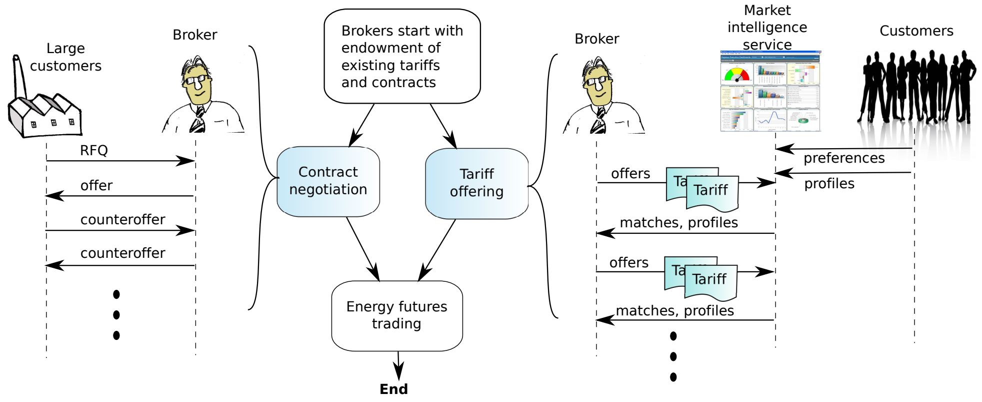


Fig. 5.7. Contracting process. Tariff offerings proceed in parallel with individual contract negotiation.

Contract and tariff terms and conditions must be described in a language that has clear semantics along with the necessary features to describe a variety of possible business agreements between brokers and their customers. The development of a common semantic model and a common pricing model to describe various kind of energy tariffs are considered top priorities on the EPRI / NIST Smart Grid roadmap for the development of a smart grid (von Dollen, 2009). With no common standard in place to build on for TAC Energy, the work of Tamma et al. (2005) serves as starting point. They developed an ontology that describes a negotiation process including (i) the involved parties, (ii) the object to negotiate on, and (iii) the negotiation process, i.e. the economic mechanism itself.

Within the TAC Energy domain, negotiations and the contracts (including tariffs) that are the subject and result of negotiations must be able to specify:

Time: including points in time, time intervals, periodicity (days, weeks, months, etc.), and temporal relationships (before, after, during, etc.). These terms can be used to specify contract duration as well as other time-related contract terms.

Energy: including amounts of energy produced or consumed, and rate of production or consumption (power). Some contracts or tariffs will also need to specify amounts of power that can be remotely controlled, for example by shutting off a domestic water heater for 15 minutes every hour during peak demand periods. Such remotely-controllable sources or loads are called "balancing power."

Money: Agreements must specify payments to or from the customer based on time (one-time signup fee or bonus, fixed monthly distribution fees), or time and energy (fixed or variable prices for a kilowatt-hour).

Communication: contract award and termination, notification of price changes, etc.

During the contracting phase, a broker must use tariff offerings and contract negotiations to develop a portfolio of contracted consumers and producers. To do this, brokers will need to estimate and reason about consumer and producer preferences in order to design appropriate tariffs and to appropriately respond to Requests for Quotes (RFQs). Brokers will also need to estimate future consumer and producer behavior to build a portfolio that has well-balanced demand and supply over time and that provides sufficient balancing capacity to achieve an acceptably low risk of execution-time imbalance.

Commonly, companies delegate the tasks of determining customer preferences and estimating business potential for new products (tariffs) to their marketing departments, or they outsource them to specialized service providers. Within the competition scenario, brokers may request such information from the *market intelligence* service (c.f. Fig. 5.7). This service is used somewhat differently for developing tariffs and individual contracts, as described in the next subsection.

The market intelligence service also provides brokers with historic consumption time series for all consumers and producers already under contract. With these time series at hand, a broker will be able to estimate how much generation and consumption capacity will be available over time and whether its portfolio is well balanced. Figure 5.8 shows an example of such a historic time series for a wind turbine with a nominal capacity of 150kW.

Negotiating tariffs

Tariffs are offered contracts that can be accepted or not by anonymous energy consumers and producers. The problem faced by broker agents in a competitive market is how to know whether a particular tariff will "sell." What happens in the real world is that firms are continually bidding against each other, attempting to attract the most "desirable" customers with their offerings.

One way to simulate this process is to allow brokers to offer tariffs in multiple "rounds," with the number of rounds $|\mathcal{R}|$ indeterminate to prevent "sniping" attacks. In each round $r \in \mathcal{R}$, agents are permitted to add or withdraw tariffs from their current offerings, resulting in a set of tariffs $\mathcal{U}_r$ for round r. The market intelligence service then runs a customer preference model (under development) to allocate customers to offered tariffs. Each agent is then provided with the number of customers who would agree to each of their offered tariffs, and they may then query the market intelligence service for predicted "demand profiles" for the projected customer base associated with each of their currently offered tariffs. These are simply

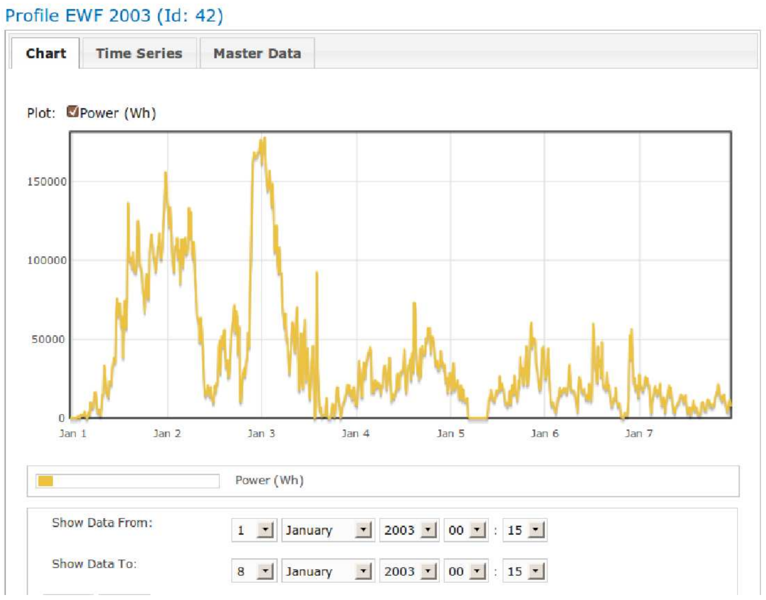


Fig. 5.8. Sample wind turbine generation timeline as provided by the market intelligence service.

aggregated time series for the set of customers who currently prefer the individual tariffs. At the end of the last round, no more offerings may be made, and brokers are charged a fee for each concurrently offered tariff. In other words, in each round r, a set of tariffs $\mathcal{U}_{b,r}$ is offered by broker b. If the fee for offering a tariff is p^{fee}, then the total tariff fee p_b^{fee} for broker b in the current contracting cycle will be

$$p_b^{\text{fee}} = p^{\text{fee}} \max_{\mathcal{R}}(|\mathcal{U}_{b,r}|), \forall r \in \mathcal{R}. \tag{5.1}$$

At the end of the last round, all currently-offered tariffs will be available for inspection by all agents through the market intelligence service.

The customer preference model aggregates a realistic range of household and business models. The function of the market intelligence service is not to “clear” the market in any sense, but rather to simply reflect the aggregate behavior of a population of agents with a range of preferences. As in the real world, individual customers will act in their own interest, even if that interest means that they are not paying attention to the cost of their electrical power at any particular time. In other words, every customer always has a tariff that he had agreed to in the past, and new tariff offerings from brokers may fail to attract the attention of most customers. This will protect the tariff market from large swings, and will prevent a single broker from cornering the market easily.

Negotiating individual contracts

Individual contracts are negotiated through an RFQ process, initiated by producers or consumers of power, and proceeding through one or more rounds with as many agents as continue to be interested. The process ends when any party accepts the current contract, or when either the RFQ originator or all brokers choose to withdraw. The smallest entities that will engage in this process will have capacities of at least 100 times the mean demand of individual households. The specifics of the negotiation process are undefined at this point, but there are many examples in the literature, such as Jonker and Treur (2001).

Execution phase

During an execution phase (see Figure 5.9), each broker must manage the supply and demand resources acquired during the contracting phase over a set of at least seven consecutive simulated days. Besides strong diurnal effects energy demand also differs significantly between working days and weekends. The length of seven days ensures an inclusion of both type of days within each execution phase. The exact length of an execution phase is drawn from a random distribution but is not revealed in advance to the agents, to reduce boundary effects within the competition.

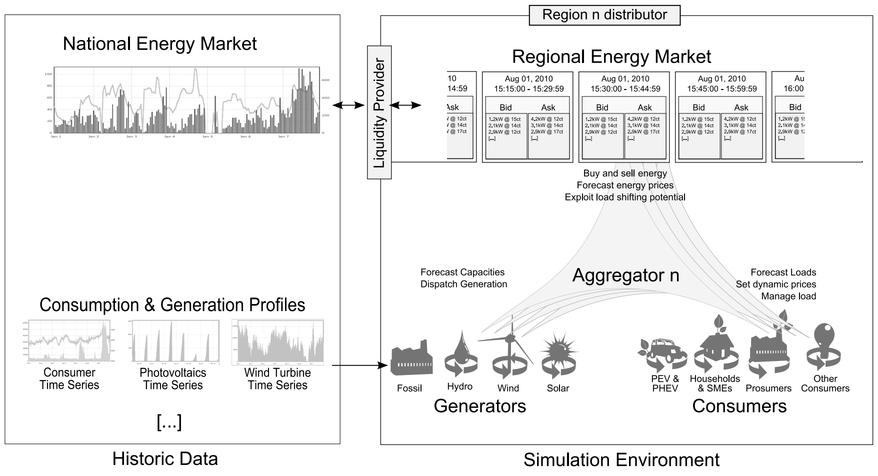


Fig. 5.9. Execution phase

The broker's main task during this timespan is to balance his customer and producer portfolios. The broker needs to ensure that the total energy demand and supply of the consumers and producers in his portfolio are balanced at any given point in time throughout the whole execution phase. Deviations between production and consumption might still occur but will be charged an (expensive) balancing power fee. The total energy consumption for broker b in timeslot s is

$$e_c(b,s) = e_{\text{ex}}(b,s) + \sum_{i=1}^{|\mathcal{C}_b|} e_i(s) \tag{5.2}$$

or the sum of the loads during timeslot s of each energy consumer in the set $\mathcal{C}_b$, the consumers in the portfolio of broker b, plus the energy exported from the grid by broker b during timeslot s through sales commitments in the regional energy market. Similarly, the total energy production for broker b in timeslot s is

$$e_g(b,s) = e_{\mathrm{im}}(b,s) + \sum_{j=1}^{|\mathcal{G}_b|} e_j(s) \tag{5.3}$$

or the sum of outputs during timeslot s of each energy producer in the set $\mathcal{G}_b$ of producers in the portfolio of broker b, plus the energy imported by b through purchase commitments in the regional energy market.

In this context, balance between supply and demand means that supply equals demand for each broker in each timeslot,

$$\forall s \in \mathcal{S},\ e_g(b,s) - e_c(b,s) = 0 \tag{5.4}$$

Note that $e_g(b,s)$ can include an arbitrary portion of contracted balancing power, and $e_c(b,s)$ may include an arbitrary portion of contracted controllable load as described in the following subsection. Ultimately, it is the job of the Independent System Operator (ISO, part of the simulation) to ensure exact balance between supply and demand in real time. Any imbalance remaining after summing supply and demand across all brokers will be balanced by the ISO using its own resources (for example, it could start up a gas turbine) and charged to the brokers who are responsible for the residual imbalance.

A broker's consumer and producer portfolio (i.e. the set of contracts) remains stable throughout an execution phase, but the overall energy demand and supply within the portfolio is volatile over time. The reason is that the actual behavior of producers and consumers is modeled based on historic generation or consumption profiles of corresponding real world entities. A wind turbine under contract with a broker will be bootstrapped with a historic time series data of a real wind turbine. Figure 5.8 shows an example for such a time series. The wind turbine modeled within the competition will expose the same significant generation volatility over time as its real world counterpart did; the same applies to consumers.

Collecting information and predicting the future

As during the contracting phase, a broker may request historic time series data for the seven preceding days (i.e. the approximate length of one complete execution phase) from the market intelligence service for all producers and consumers currently under contract. With this historic data series at hand a broker will be able to build its own prediction model for future energy consumption and production of its portfolio, in order to be able to detect and address likely future imbalances.

In general, retrieval of future time series data from the market intelligence service is not permitted throughout the competition with one exception: For intermittent generators such as photovoltaics or wind turbines, the estimation of future output solely based on historic time series data is problematic and unrealistic. Predictions about the output of these types of generators are usually based on weather forecasts as described for example in Sanchez (2006).

In order to shield brokers from having to model weather forecasts, and also because forecasts for specific generators as input for the competition are usually not publicly available[6] the approach for this competition is to permit future time series data lookups (only) for *intermittent* generators. Still, in order to model reality closely, these future time series will be artificially distorted (see Ahlert and Block, 2010, for background).

Instead of returning the exact future energy production time series for a generator j, the simulation will return an artificial (forecasted) production $e'_g(j, s_k)$ for each time slot s_k, $(n+1) \leq k \leq \eta$ from the currently running time slot s_n up to a future time slot s_η. Each forecasted generation capacity $e'_g(j, s_k)$ is calculated as:

$$e'_g(j, s_k) = e_g(j, s_k)\,(1 + \zeta_g(j, s_k))\ ,\ s_n \prec s_k \preceq s_\eta \tag{5.5}$$

where $\zeta_g(j, s_k)$ represents a forecasting error for the generation capacity of generator j in time slot s_j. For example, if $\zeta_g(j, s_k) = 0.05$ for generator j during timeslot s_k, then the prediction will be 5% higher than the actual output. The forecasting error $\zeta_j(j, s_k)$ is computed using a stochastic process, in particular a Wiener process (Orey and Pruitt, 1973).

$$\zeta_j(j, s_k) = \varepsilon_k + \alpha \sum_{i=n+1}^{k-1} \varepsilon_i \tag{5.6}$$

Variable α is used to adjust the autocorrelation between two forecasts for generation capacity in adjacent time slots. Setting $\alpha = 1$ results in a perfect positively autocorrelated time series while $\alpha = 0$ leads to a completely uncorrelated time series. In order to produce realistic forecasts, α needs to be adjusted manually for each time series (each type of intermittent generator) for which to create forecasts. The noise variable $\varepsilon_i(n+1 \leq i \leq k-1)$ is normally distributed $\varepsilon_i \sim N(0, \sigma_i)$ and the standard deviation σ_i is defined as:

$$\sigma_i = \phi_1 \cdot \frac{\sqrt{2\pi}}{2} \cdot \frac{(\eta + i\frac{\phi_\eta}{\phi_1} - i - \frac{\phi_\eta}{\phi_1})}{\sqrt{1 + \alpha^2 \cdot i - \alpha^2 \cdot (\eta - 1)}} \tag{5.7}$$

The result of this process is that predictions of the future become progressively less accurate as the time horizon increases. This process can be modeled by considering the mean absolute percentage error (MAPE) of a prediction. ϕ_1 is assumed to be the mean absolute percentage error (MAPE) of a production forecast in the first forecasted time slot (s_{n+1}), and ϕ_η is the MAPE for production in the last forecasted time slot s_η. The expected errors $\phi(s_k)$ for the time slots s_k, $(n+1) < k < \eta$ are linearly interpolated by this process:

$$\phi(s_k) = \phi_1 + (\phi_\eta - \phi_1)\frac{k-1}{\eta - 1} \tag{5.8}$$

The idea for the creation of this type of artificial forecasts originally stems from Ahlert and Block (2010) and is described there in detail.

Based on (i) the historic generation schedules of "predictable" generators (e.g. micro turbines or CHP plants), (ii) the historic consumption schedules of the consumers

[6] Such forecasts are provided by specialized companies that charge significant fees.

under contract, and (iii) the forecasted generation schedules of intermittent generators, a broker will have to predict the estimated future energy generation and consumption schedules for its portfolio as visualized in Figure 5.10. Note how the uncertainty for the expected demand and supply in time slots far into the future is higher than for those near to the current time slot. In Figure 5.10(a) the maximum expected overall generation capacity for broker b in time slot s_{n+5}, $e'_g(b, s_{n+5})$, is much lower than the expected overall load $e'_c(b, s_{n+5})$. But as the MAPEs for both numbers are high (indicated as gray boxes), the accuracy of this prediction is low.[7] 120 Minutes later (Figure 5.10(b)) the expectation values for supply and demand remained unchanged but the standard deviation decreased. At this point in time the broker is able to predict an excess demand situation (e.g. see Ketter et al., 2009) for the time slot with a good confidence and thus can now already introduce appropriate countermeasures. In this case he decided to acquire additional energy for time slot s_{n+5} from the regional energy exchange as indicated in the Figure. An alternative would be to adjust energy and supply within b's portfolio as described in the following section.

Adjusting energy demand and supply

For each time slot s, each broker b must balance expected supply and demand closely enough that the ISO can achieve exact balance without expending any of its own resources. Expected demand is the total expected load, or the sum of committed power exports and the expected loads $e'_i(s)$ of each consumer i in the broker's consumer portfolio $\mathcal{C}_b$ during time slot s (see Equation 5.2):

$$e'_c(b, s) = e_{\mathrm{ex}}(b, s) + \sum_{i=1}^{|\mathcal{C}_b|} e'_i(s) \tag{5.9}$$

Expected supply is committed power imports plus total expected production capacity of all generators g within the broker's portfolio $\mathcal{G}_b$ during timeslot s (see Equation 5.3):

$$e'_g(b, s) = e_{\mathrm{im}}(b, s) + \sum_{j=1}^{|\mathcal{G}_b|} e'_j(s) \tag{5.10}$$

These values are maximum values in case some consumers and/or some producers in the broker's portfolios have agreed to external control, presumably in exchange for better prices. For example, a combined heat and power generator with a nominal output of 50kW can be adjusted by an external control so that its real production is within certain boundaries, e.g. [40kW – 50kW]. Similarly, a domestic water heater may be configured to permit remote shutoff for up to 15 minutes every hour. The

[7] Note that the MAPE for the overall consumption stems from the demand forecasting model the broker has to build on its own. The MAPE for the overall generation stems in part from the artificial distortion of future generation data as described in this section and in part from a generation prediction model that the broker has to implement for forecasting its non intermittent generation capacities like e.g. CHP engines or micro gas turbines

total controllable load for a broker b during timeslot s is $\epsilon_c(b,s)$, and the total controllable production capacity is $\epsilon_g(b,s)$. As long as $e_g(b,s) - \epsilon_g(b,s) \leq e_c(b,s)$ and $e_c(b,s)-\epsilon_c(b,s) \leq e_g(b,s)$, then supply and demand during timeslot s is expected to be in balance. Within this range, the ISO will either reduce load or reduce output as needed to achieve exact balance.

The dispatching of balancing power (or load) by the ISO is done only during the current simulation time slot s_n. In Figure 5.10(a), one can see in the current slot s_n that both the actual observed supply and demand have deviated from the forecasted overall supply and demand for broker b. But as the difference between $e_c(b,s_n)$ and $e_g(b,s_n)$ was smaller than $\epsilon_g(b,s_n)$, the controllable production capacity of broker b in this slot, the ISO was able to automatically reduce supply such that overall demand and supply for timeslot s_n was rebalanced.

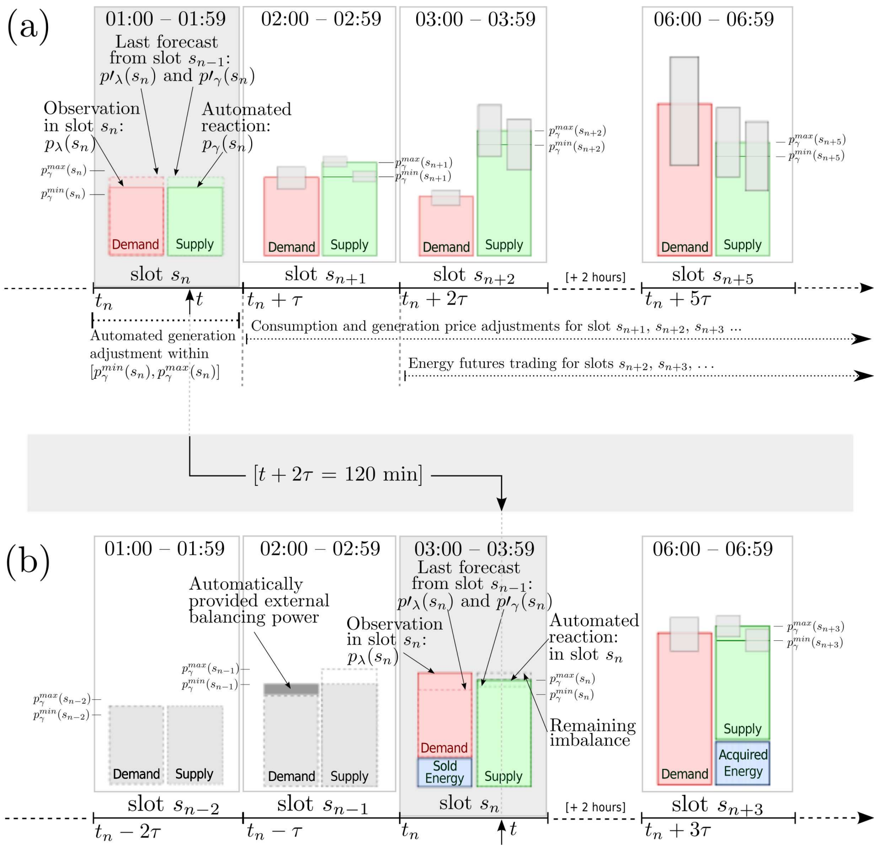

(a)
01:00 – 01:59
Last forecast from slot s_{n-1}: $p\prime_\lambda(s_n)$ and $p\prime_\gamma(s_n)$
Observation in slot s_n: $p_\lambda(s_n)$
Automated reaction: $p_\gamma(s_n)$
$p_\gamma^{max}(s_n)$
$p_\gamma^{min}(s_n)$
Demand
Supply
02:00 – 02:59
03:00 – 03:59
06:00 – 06:59
$p_\gamma^{max}(s_{n+1})$
$p_\gamma^{min}(s_{n+1})$
$p_\gamma^{max}(s_{n+2})$
$p_\gamma^{min}(s_{n+2})$
$p_\gamma^{max}(s_{n+5})$
$p_\gamma^{min}(s_{n+5})$
slot s_n
slot s_{n+1}
slot s_{n+2}
[+ 2 hours]
slot s_{n+5}
t_n
t
$t_n+\tau$
$t_n+2\tau$
$t_n+5\tau$
Automated generation adjustment within $[p_\gamma^{min}(s_n), p_\gamma^{max}(s_n)]$
Consumption and generation price adjustments for slot s_{n+1}, s_{n+2}, s_{n+3} ...
Energy futures trading for slots s_{n+2}, s_{n+3}, ...
$[t+2\tau = 120$ min$]$
(b)
01:00 – 01:59
02:00 – 02:59
03:00 – 03:59
06:00 – 06:59
Automatically provided external balancing power
$p_\gamma^{max}(s_{n-2})$
$p_\gamma^{min}(s_{n-2})$
$p_\gamma^{max}(s_{n-1})$
$p_\gamma^{min}(s_{n-1})$
Last forecast from slot s_{n-1}: $p\prime_\lambda(s_n)$ and $p\prime_\gamma(s_n)$
Observation in slot s_n: $p_\lambda(s_n)$
Automated reaction: in slot s_n
$p_\gamma^{max}(s_n)$
$p_\gamma^{min}(s_n)$
Remaining imbalance
Sold Energy
Acquired Energy
$p_\gamma^{max}(s_{n+3})$
$p_\gamma^{min}(s_{n+3})$
slot s_{n-2}
slot s_{n-1}
slot s_n
slot s_{n+3}
$t_n-2\tau$
$t_n-\tau$
t_n
$t_n+3\tau$

Fig. 5.10. Broker's expected and actual energy supply and demand at two points in time.

For time slot s_{n+1} in Figure 5.10(a), expected overall demand is forecasted to be within range of the available production capacity, but the uncertainty envelope (grey boxes) shows that this is not certain. In other words $e'_g(b,s_{n+1}) - \epsilon_g(b,s_{n+1}) \leq e'_c(b,s_{n+1})$. After 2τ simulation time has elapsed (Figure 5.10(b)), this slot is now designated s_{n-1}, and one can see that the real consumption $e_c(b,s_{n-1})$ in this time slot turned out to be lower than $e_g(b, s_{n-1}) - \epsilon_g(b, s_{n-1})$. This means that even after the simulation environment reduced the broker's production capacity to its minimum level, the overall production still exceeded the overall consumption. In this case the ISO used an external balancing load (either a shortage of power from some other broker, or something like a large pumped-storage power plant outside the broker's portfolio) to absorb the excess generated energy. In the energy industry this type of balancing power is usually called an "ancillary service" and its utilization is billed to the broker at a defined (high) price.

In slot s_{n+2} in Figure 5.10(a), a significant difference between overall production and overall consumption is forecast. Internal balancing capacity is likely to be insufficient for leveling the expected difference. In order to avoid the (expensive) utilization of external balancing power, broker b can either sell some of its surplus energy on the regional energy exchange market, or use its contracted pricing power to try to encourage (i) some or all of its consumers to increase their demand, or (ii) some or all of its producers to reduce their production. Technical adjustments (e.g. a remote activation of loads at consumer premises) is not allowed within the competition. But a consumer's energy consumption is subject to the energy consumption price for consumer i in a time slot s, which is defined as $p_c(i, s)$. In particular,

$$\hat{e}_c(i, s_{n+2}) = e'_c(i, s_{n+2}, p_c(i, s_{n+2})) \tag{5.11}$$

is defined as the predicted load for consumer i in time slot s_{n+2}, given price $p_c(i, s_{n+2})$. If the broker changes the underlying consumption price to $p'_c(i, s_{n+2})$ the forecasted consumption of this consumer is expected to increase as

$$\hat{e}'_c(i, s_{n+2}) = e'_c(i, s_{n+2}, p'_c(i, s_{n+2})) \tag{5.12}$$

The ratio of demand change to price change

$$PE_i = \frac{\hat{e}_c(i, s, p) - \hat{e}_c(i, s, p')}{p - p'} \tag{5.13}$$

is called the "price elasticity" for consumer i. Price elasticities will have to be modeled within the different consumer agents provided by the competition environment following empirical findings on price elasticity as described for example in Spees and Lave (2008); Siddiqui et al. (2004).

Some producers in the broker's portfolio (such as electric vehicle batteries that can be discharged into the grid) might have agreed to flexible pricing as well, and therefore their output will be sensitive to price in a similar way. In other words, the power generation capacity of broker b in time slot s, $e_g(b, s)$, is likely to change if the generation price $p_g(j, s)$ is changed to $p'_g(j, s)$, decreasing if $p'_g(j, s) < p_g(j, s)$.

Different kinds of customer models are available within the TAC Energy simulation framework, such as electric vehicles, CHPs, wind turbines, and private households. As an example Figure 5.11 shows the difference between the electricity load profile of a household's washing machine under a flat tariff and under real-time pricing. A simulation tool generates the load profile of a household for one year under a flat tariff by using historical data. For shifting it is assumed that customers face real-time electricity prices. The simulation shifts the use of each device to the cheapest time slot within a day under consideration of customer preferences.

For estimating the load profile of a household the consumption share of a device on the total yearly electricity consumption of a four person household is used. Historical mean values for annual electricity consumption and the share of a device on this consumption are used. Yearly load per device and the consumption of a device enable to calculate the number of runs for each device. The runs are distributed on different time slots of a year. Each season, day of the week and time of the day has different probabilities for the allocation of device runs. Additionally, the presence of the persons living in the household is determined and for some devices (e.g., washing machine, dish washer) the calculated start slot for a run is updated if no one is at home.

In the shifting simulation for the washing machine shown above it is assumed that a household loads its washing machine in the morning between 5:30 and 7:30, for all machines started after 5:30 am. For a wash that started between 0 am and 5:30 am under the flat tariff the point of loading does not change. After loading the machine the customer selects a time period in which the wash should finish. There are four modes available: finish within the next 5 hours, between 5 to 10 hours from now, after 10 hours from now or the wash can be done at each time slot during that day. Selection of a mode is based on the presence of the household occupants. A mode can only be selected if a customer is at home during the two hours after the end of the interval specified by a mode. Next, the simulation determines the cheapest period for running the washing machine, which must be within the interval set by the selected mode. 66% of the washes have to be finished before 10:30pm as 66% of the German population are asleep at 11pm (Meier, 2004).

Figure 5.11 sums the load corresponding to a washing machine in each hour during the whole year under a flat tariff and under real-time pricing. The average European Energy Exchange (EEX) price curve explains the differences between the two load curves: in times of high prices load is reduced, meanwhile a load increase in low period prices can be observed.

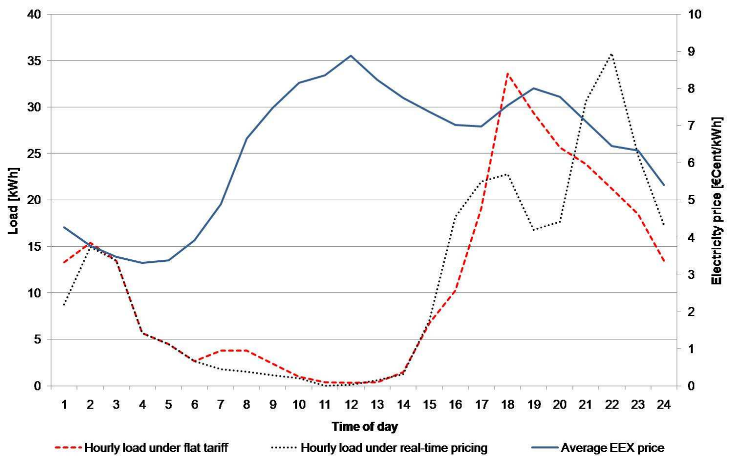


Fig. 5.11. Demand shifting of a household washing machine.

Buying or selling futures on the energy market

The re-adjustment of energy prices for consumers and producers as well as the advance reservation of (partial) producer capacity as balancing power reserves are two possibilities to level out a broker's portfolio over time. Besides these two options, a third one is to buy missing or to sell excess capacities on the energy market. Within the competition this market is modeled as a continuous double auction with uniform pricing and thus resembles the prevalent mechanism design in place for energy spot market trading in Europe and North America (Meeus and Belmans, 2007). On this market standardized energy futures are traded. An energy future is a binding commitment to consume or to produce a defined amount of energy (e.g. 1kWh) within a defined future time slot (e.g. Aug 01, 2007, 03:00 – 03:59) at a defined price (e.g. 20 ct/kWh). In order to buy or sell energy futures, a market participant sends bid or ask orders to the energy market, which then clears (matches) all incoming bids in a continuous double auction.

Table 5.2 shows a sell order before its submission to the energy market (left) and after its clearing (right). In this example Broker 1 wants to sell 100 kWh of power within the time slot Aug 01, 2007 03:00 – 03:59 at a minimum price of 0.14 EUR/kWh. The offer is set to expire on Aug 01, 2007 01:52h, i.e. if no matching takes place until that time, the order will be canceled. At 01:37 the order is matched on the energy market at a price of 0.15 EUR/kWh and a quantity of 100kWh. Broker 1 managed to sell 100kWh of energy for this time slot, which helps him to better balance his portfolio's overall energy supply and demand in this time slot (see Figure 5.10).

(a) Submitted Sell Order

Order ID:	1
Order Owner:	Broker 1
Timeslot:	Aug 01, 2007 03:00 – 03:59
Order Type:	Sell
Order Status:	Open
Quantity:	100 kWh
Limit:	0.14 EUR / kWh
Created at:	Aug 01, 2007 01:37
Valid Until:	Aug 01, 2007 01:52
Clearing Date:	–
Clearing Quantity:	–
Clearing price:	–

(b) Matched Sell Order

Order ID:	1
Order Owner:	Broker 1
Timeslot:	Aug 01, 2007 03:00 – 03:59
Order Type:	Sell
Order Status:	Matched
Quantity:	100 kWh
Limit:	0.14 EUR / kWh
Created at:	Aug 01, 2007 01:37
Valid Until:	Aug 01, 2007 01:52
Clearing Date:	Aug 01, 2007 01:42
Clearing Quantity:	100 kWh
Clearing price:	0.15 EUR / kWh

Table 5.2. Orders submitted to and later matched on the regional energy market.

Besides the brokers trading energy futures on the regional energy market, there is also a special agent called "liquidity provider." The liquidity provider is basically representing the point of common coupling (PCC) between simulated region and national grid on the regional market. He can buy energy at the national market and transfer it via the PCC to the simulated region and vice versa. Thus the liquidity provider serves as an arbitrage agent that levels prices of the regional and the national energy market and constitutes an *explicit market coupling* (Meeus and Belmans, 2007).

Agent decision problems

Brokers in the simulation may be individual autonomous learning agents, as in TAC SCM (Collins et al., 2005; Ketter et al., 2009). They may also be agent communities; for example, it might make sense to separate the contracting and execution behaviors into two separate agents. It is also possible that the "agent," at least for the contracting phase, could be one or more human decision-makers operating with the assistance of appropriate user interfaces and decision-support tools. This section describes in some detail the decisions such broker agents must make to operate within the TAC Energy simulation. The discussion is separated into the Contracting and Execution phases as described in the previous sections. There is very little overlap between the behaviors and decisions during these two phases, except that (i) both phases allow trading in the regional spot market, and (ii) the decisions made during the contracting phase will profoundly influence the required behaviors during the following execution phase.

Contracting phase

The primary goal of the contracting phase is to acquire access to power sources and customers that result in a portfolio that is profitable and balanced, at least in expectation, over the period of the next execution phase. A secondary goal is to manage financial and supply/demand imbalance risks. For example, an agent will benefit from having reasonably-priced energy sources that can be expected to produce power when demand is expected to be highest within its load portfolio. Predictability is also important, and will generally improve both with volume (because noise as a proportion of demand or supply will be lower with larger numbers of randomly-behaving sources and load, even if they are correlated) and with a balanced portfolio of uncorrelated sources and loads. Risk can be managed by acquiring uncorrelated sources and loads that can be expected to balance each other in real time, by acquiring storage capacity, by acquiring sources that can be used as needed (balancing sources), and by trading futures contracts on the regional exchange.

At the beginning of a contracting phase, an agent will have some number of contracts in force, having negotiated them earlier. Such contracts have expiration dates beyond the current date. Also, tariffs offered earlier may remain in force; customers who have agreed to a tariff in the past may or may not have an opportunity to opt out and choose a different tariff, and if they have the opportunity they may not choose to exercise it.

Acquire power sources

Power source commitments are obtained by three different methods:

- Large local sources (large wind turbines, wind farms, large CHP plants, etc.) are traded in the local market through the RFQ process.
- Small local sources (household and small-business sources) are obtained by offering tariffs in the local market.

- Power from the regional grid is obtained by trading in the regional exchange.

Power sources can be continuous or intermittent, and local continuous sources may have a non-zero balancing component. Continuous sources include power obtained from the regional exchange, as well as the continuous portion of the output from many CHP and hydro plants. Intermittent sources include most renewable sources such as wind and solar plants.

Acquire storage capacity

Storage capacity can be used to absorb excess power or to source power during times of shortage. Power can be absorbed by capacity that is not fully charged, and sourced by capacity that is above its contracted minimum charge level. Storage capacity that is below its minimum charge level is considered to be a load that is possibly responsive to real-time price signals.

Storage capacity can be contracted through the local market through the tariff or the RFQ process. For example, individual owners of PEVs could sign up for tariffs that provide for both charging of the batteries as well as limited discharging as needed for load balancing by the contracted broker. On the other hand, a battery-exchange service for electric vehicles might negotiate a contract for the use of a portion of its current battery inventory for balancing purposes.

Acquire loads

Loads may be contracted through both the local market and the regional exchange, as is the case with power sources.

- Large local loads (industrial facilities and large office parks, for example), could negotiate rates through the RFQ process.
- Small local loads (households and small businesses, for example) must choose tariffs in the local market.
- Agents may choose to sell future power capacity in the regional exchange for periods when it expects to have a surplus. Such advance sales are binding commitments; the sold quantity of power will be transferred out of the system during that interval at a constant rate.

Execution phase

A detailed timeline of events prior to and during the execution phase is shown in Figure 5.12. At time t, the simulation is partway through one of the 60-minute timeslots defined by the regional exchange, which started in the past at time t_n. Trading has closed on that slot, and on slot s_{n+1}. Trading will close on slot s_{n+2} at the beginning of slot s_{n+1} at time $t_n + 60$. Between time t and time $t_n + 60$, the agent may continue to trade in slot s_{n+2} and all future slots. The agent may also send price signals to its contracted loads and sources at any time, to the extent allowed in their respective contracts. At the end of each slot (time $t_n + 60$ in this example), the agent will receive information about its supply and demand status at the end of the just-completed slot (slot s_n), and may then send price signals to its contracted sources and loads. These signals will arrive during the next timeslot (slot s_{n+1} in this example) but will not take effect, regardless of the contract terms, until the beginning of the following timeslot (slot s_{n+2}).

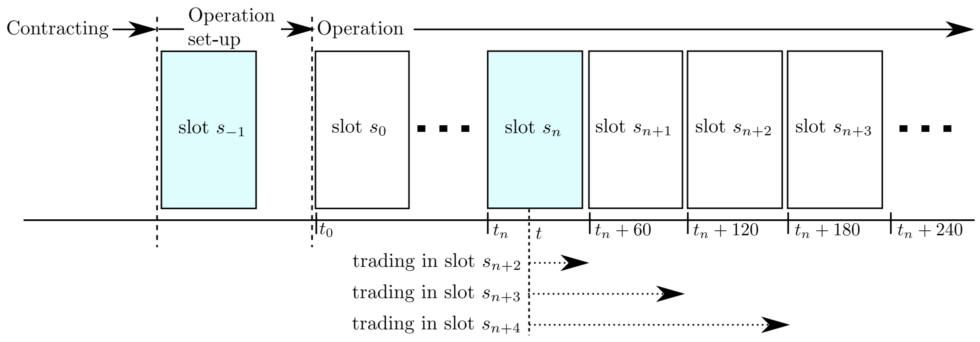


Fig. 5.12. Timeline of interaction between agent and simulation.

Execution set-up

At the end of the contracting phase, the agent has knowledge of its current contract commitments, and of the number of customers who have agreed to its offered tariffs. Execution covers some period of time while these contracts are in place. In order to avoid a lengthy period of adjustment at the beginning of each execution phase, a single timeslot s_{-1} is run in execution mode immediately prior to the period that is to be simulated and evaluated. Results of this preliminary timeslot, along with the current date and time, are then made available to the agent, and the agent is given an opportunity to request history and forecast data, to adjust its variable prices, and to trade in the regional exchange, before full execution commences. During this setup interval, energy can be traded for all timeslots starting with s_0 instead of being restricted to timeslots starting with s_1, and leadtime restrictions for price changes are waived. This allows the execution phase to begin in a relatively stable state, without requiring agents to spend multiple cycles fine-tuning their balance.

Execution

After the setup period, beginning at time t_0, the simulation runs continuously. Agents may trade in the exchange and set variable prices at any time. At the end of each timeslot, the agent will receive a performance report giving the supply and demand volume for each of its contracts and tariffs. For each future timeslot $s \succ s_n$ a broker b must maintain a forecast of its total expected load and capacity. Total load is the sum of the expected loads $e'_c(i, s)$ of each consumer i in the broker b's consumer portfolio

$\mathcal{C}_b$ for timeslot s. Total production is the sum of expected capacity $e'g(j,s)$ for each producer j within its producer portfolio $\mathcal{G}_b$. Given this information, the agent's task is to adjust prices, and trade in the regional exchange, in order to achieve expected balance.

Performance evaluation

Within a competition the performance of its participants has to be evaluated and compared at a certain point in time. This is usually accomplished by rank ordering all participants according to one or more defined performance criteria and to declare the best performer in this rank order winner of the competition. This principle also applies to TAC Energy. Consequently this section describes (i) the performance criteria used to rank order the TAC Energy participants, and (ii) the sampling method.

Performance criteria

Each TAC Energy participant (broker) is assessed and an overall rank order of all participants is created based on overall profits p^{profit}, calculated as the (monetary) payments, p^{pay}, minus costs, p^{cost}, minus fees, p^{fee}:

$$p^{\text{profit}} = p^{pay} - p^{\text{cost}} - p^{\text{fee}} \tag{5.14}$$

- **Payments** are monetary transfers from consumers to brokers and are based on the agreed contract conditions and the actual (ex-post) measured energy consumptions of the respective consumers.
- **Costs** are monetary transfers from brokers to producers and are based on the agreed contract conditions between the respective producer and broker and the actual (ex-post measured) energy produced.
- **Fees** are (i) the cost for external balancing power used, and (ii) a carbon tax. The carbon tax is a fixed fee (in EUR / kWh) for each kWh of energy produced from non renewable energy sources. The carbon tax remains constant throughout a competition and is publicly announced ahead of the start of the first round.

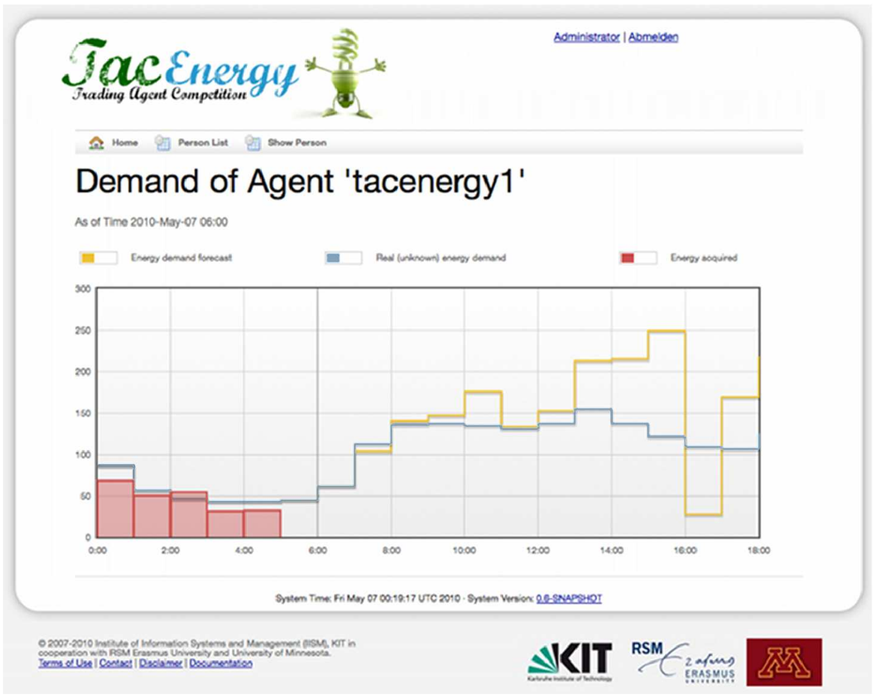
Sampling method

Several randomly chosen timeslots from each each execution phase are selected as "reference timeslots" before a particular competition starts. The chosen timeslots for a particular execution phase are kept secret until that phase ends. Afterwards, profits are calculated for each of the reference timeslots of the particular execution phase and then averaged over all reference timeslots. The individual profits of the reference timeslots as well as the average profit of an execution phase are publicly announced immediately after the phase ends. The overall ranking of brokers is calculated as the average profit over all reference timeslots from all execution phases. These will be ranked to produce the *winner ranking* at the end of the competition.

Initial results from prototype

At the time of writing of this thesis, an initial version of the competition platform and a demo agent for the execution phase of the TAC Energy competition is implemented and publicly accessible at `http://www.tacenergy.org`. This version of the competition is not yet feature complete in terms of the competition scenario described before, but merely represents an intermediate state of the development.

In its current version, broker agents are assigned a fixed portfolio of energy sources and loads to manage, and they must sell or acquire energy on the exchange in order to achieve balance. Price sensitive-consumers and producers are not implemented yet. The screenshot in Figure 5.13 shows the view of one agent at just before 6:00. At this point, one can see that the agent purchased less than the needed power for timeslots 0:00, 1:00, 3:00, and 4:00, and more than needed in the 2:00 timeslot.



Demand of Agent 'tacenergy1'

As of Time 2010-May-07 06:00

System Time: Fri May 07 00:19:17 UTC 2010 · System Version: 0.6-SNAPSHOT

© 2007-2010 Institute of Information Systems and Management (ISM), KIT in cooperation with RSM Erasmus University and University of Minnesota.

Fig. 5.13. One agent's view during an execution phase.

The problem the agent must solve is illustrated by the difference between the "forecast" and "demand" curves for the future. The agent sees only the forecast data, of course; the TAC Energy competition platform produces the forecast from the actual supply and demand data for the agent's portfolio using real historic consumption data from the time series data store (see Section 4.2.3) as the basis, which is then artificially distorted as described in Ahlert and Block (2010). Given these forecasts, the agent must acquire (or sell) enough energy, by trading in future timeslots, to achieve balance before each timeslot becomes the current timeslot.

This version of the TAC Energy platform is fully functional and open to interested participants. Like this, early "hands-on" feedback can be elicited from these participants through an online feedback elicitation tool as shown in Figure 4.7 in Section 4.1.3. This feedback is used to continuously revise and refine the TAC Energy requirements, which the project team notes in the form of user stories as shown in Figure 4.2, and collects and prioritizes them on a taskboard as shown in Figure 4.4. In short, the development of the TAC Energy platform has successfully been conducted so far according to the agile market engineering process model described in Chapter 4.

Technically, the TAC Energy market simulation platform is based on (i) the cda market template (see Section 4.2.2) embedded into the market simulation framework described in Section 4.2.3.

The market intelligence service (c.f. Figure 5.7) is realized as an extension to the time series data store component of the market simulation framework (see Section 4.2.3). Also the liquidity provider agent, used for coupling the regional spot market to the national energy exchange, is created based on default functionality provided by the market simulation framework. The national energy exchange market is simulated based on historic time series data stored in and provided by the time series data store component of the market simulation framework. Overall, the development of the TAC Energy platform up to its current state required an effort of about 60 FTE days for realization.

At the time of writing of this theses a seminar at Institute of Information Systems and Management with 16 graduate students is running who are building broker agents and will compete against each other in an internal competition by the end of the summer. These initial results as well as their feedback serves as further input for the finalization of the competition specification and implementation so that a full competition system can be introduced as planned in Spring 2011.

Summary of TAC Energy Use Case

The TAC Energy project is aimed at developing a competitive simulation of a market-based management structure for a regional energy grid that would closely model reality by bootstrapping the simulation environment with real historic load, generation, weather, and consumer preference and usage data. Such a simulation environment would challenge research teams from around the world to write autonomous agents, or agent-assisted decision support systems for human operators Varga et al. (1994), that could operate effectively and profitably in direct competition with each other, while also continuously balancing supply and demand from their portfolios. Teams would also be challenged to exploit the structure of the market, and that structure would be adjusted periodically to defeat counterproductive strategic behaviors. The result would be a body of valuable research data, along with a much higher degree of confidence that such a mechanism could be safely introduced into operating energy systems.

Agents in this market simulation would act as "brokers," purchasing power from distributed sources and from regional energy exchanges, and selling power to consumers and exchanges. These agents must solve a set of complex supply-chain problems in which the product is infinitely perishable, and the environment is subject to high variability and uncertainty (e.g. weather effects, equipment and network outages) and limited visibility. They will operate in a dynamic network at multiple timescales, from negotiating long-term contracts with energy producers and tariffs for customers that balance supply and demand in expectation, to day-ahead spot-market trading, to real-time load balancing. They must deal with individual customers and suppliers, while at the same time aggregating the preferences of large groups of customers into market segments for tariff offerings. They must predict supply and demand over monthly, quarterly, and yearly timescales as they develop their portfolios of supplier and customer relationships, and over hours and minutes as they adjust dynamic prices and trade in the spot market in order to maintain real-time balance in the grid.

The simulation environment and broker agents are subject to high variability, uncertainty and limited visibility. This allows, for example, the study of exogenous shocks (e.g. a power plant outage) and their impact on system stability, or competition effects among broker agents. Effects of policy changes, such as taxes and incentives can be modeled and examined too. Markets are self-organizing mechanisms, and the TAC Energy environment is designed to become an effective tool for research in self-organizing networks and complex adaptive systems Miller et al. (2007).

Technically the TAC Energy development is developed based on the agile market engineering process model and uses several of its accompanying software artifacts. Like this, it was possible to develop and run an early version of the competition after about 60 FTE days in overall effort. User feedback from the running competition platform is used to continuously refine and adjust the requirements for the further development of TAC Energy. A first, fully implemented version of the competition is set to be launched in 2011.

5.7 Summary

In this chapter the agile market engineering process model is evaluated. Section 5.1 analyses the complexity of the default cda and call market templates. The main finding from this analysis is that the market templates possess an overall low complexity and are generally small and concise in terms of program code size. Each template consists of approximately 2000 lines of program code overall and thus should be easy to adapt and to extend to particular project needs. These analytical findings are confirmed in a series of case studies where the agile market engineering process model has been used for project management and where one of the market templates served as basis for the development of the respective electronic market platforms.

After the end of the EM-Stoxx market project (see Section 5.2), its electronic prediction market platform was released as new prediction market template in the market repository and subsequently served as the (enhanced) technical basis for the AKX and EIX prediction market projects described in Section 5.3 and 5.4.

In Section 5.5 a case study is described that used and extended the default call market template to implement a proof-of-concept combinatorial heat-and-power market platform as basis for energy scheduling in regional energy grids. Insights from this project lead to the development of the TAC energy competition platform described in Section 5.6. Here the agile market engineering process model and several of its software artifacts are used to develop a competitive testbed for energy market trading automation.

All of the aforementioned projects applied the agile market engineering process model for project management and were able to quickly adapt and extend one of the default market templates according to their needs. The overall required efforts for developing the respective market platforms in the different projects range between approximately 25 and 70 FTE days.[8] Compared to previously conducted market engineering projects such as the Stoccer project with an overall development effort of approximately 1.5 years FTE (see Section 3.2) for the realization of a prediction market platform, the agile market engineering approach described in this thesis has been shown to perform well.

In Section 3.4.1 and 3.4.2 a set of requirements was formulated that the agile market engineering process model in combination with its accompanying software artifacts needs to address. In the following listing these requirements are recapitulated and their adherence is assessed.

Adjustable support for common market functionality: Several different market templates provide default market functionality for one particular market mechanism each. As was shown throughout all of the case studies, these default functionalities can be adjusted and customized to the specific project needs with relatively low effort.

Authentication & Authorization: All market templates come with a default authentication and authorization module included. To increase the trustworthiness of the claimed user identities, OpenID and LDAP, two common network authentication protocols, are supported by default in all of the market templates so that the task of reliably authenticating market participants can be outsourced to trusted third parties if required.

Integrated & adjustable information services: All market templates are equipped with default web interfaces that provide (automatically updated) market information to users. These interfaces can be adjusted and extended (see e.g. Figure 5.3) to the particular project needs with relatively little effort.

Support for multi-channel market access: By default, all market templates are equipped with a web interface and additionally provide a web services interface via the Simple Object Access Protocol (SOAP) for machine-to-machine communication. Like this, human market participants as well as, for example, electronic trading agents have access to the electronic market platforms by default.

Optional decision & automation support: Neither decision support nor support for automation technology (besides the aforementioned web services inter-

[8] The TAC Energy as well as the EIX Market projects are still ongoing as of writing of this thesis. Thus estimated efforts for these project represent only the efforts expended to date.

face for machine-to-machine communiction) is provided by default. Still decision support functionality can be easily added at limited extra effort (see e.g. Figure 5.4) if this is required in a specific project context. Also market automation functionality, e.g. by means of electronic trading agents can be integrated into the technical market infrastructure (see e.g. Section 5.6) though no trading automation is provided by default.

Simple Software Architecture: The complexity analysis conducted in Section 5.1 shows that the code base of the default market templates is concise and of relatively low complexity.

Reusability of markets: One example for "applied" reusability of the markets developed based on the agile market engineering process model is the EM-Stoxx market. After the end of this project, its code base was converted into a market template, published in the market repository, and subsequently used as technical code base for the AKX and the EIX market project.

Overall, the evaluation in this chapter shows that the agile market engineering process model and its accompanying software artifacts are suitable for the development of electronic market platforms. Both, process model and software artifacts, were successfully used in in five different real-life market engineering projects. In all of these projects, electronic market platforms for different, specific purposes have been developed and successfully operated. The efforts required for the realization of these projects were moderate as compared to similarly scoped projects that used different project management approaches and software tools for the development of electronic markets.

6

Summary & Future Work

6.1 Summary & Main Contributions

In this thesis an agile market engineering process model is introduced and evaluated. Different to previous market engineering process models, it does not rely on extensive and detailed upfront design and modeling of a new electronic market platform. Instead it fosters and supports short, lightweight, incremental development cycles, each delivering a fully functional and incrementally improved electronic market platform that can be tested and evaluated "hands-on" as a running system. The rationale in behind is to pragmatically address the inherent "wickedness" and complexity of market design and development by relying on frequent feedback from market participants and by following a *"flexible, iterative design-and-build strategy"* as recommended by Kambil and van Heck (2002, p. 100). Like this, a continuous improvement and change process is established that favors learning, pragmatic development and refactoring over extensive design and theoretical upfront modeling.

Several software artifacts have been developed that were designed to support and facilitate the agile market engineering process.

The *Market Design Knowledge Base* is a software platform for storage and retrieval of market (mechanism) knowledge. Its parametric approach to describing market engineering knowledge is flexible enough to preserve insights on how (or how not) to design and realize electronic market platforms optionally also covering information on the respective market environment, the traded products, or the types of market participants a specific market design recommendation is valid for.

The *Market Repository* was built to support market developers in quickly instantiating new market instances from a set of previously developed and archived market platforms. Instead of developing new markets from scratch every time, and instead of using a generic but complex and hard to extend market runtime environment, small, simple, and specific *market templates* have been developed as technical foundations to built new electronic market platforms upon. Three basic market templates, one for a continuous double auction market, one for a call market, and one specific prediction market template were developed and published in the market repository as initial market templates.

The *Market Simulation Framework* was developed to provide a convenient to use agent-based simulation environment as realistic testbed for newly developed market platforms. It allows for an extensive and realistic testing in cases where the market development team is unsure about the fundamental market design. The framework consists of computer-grid enabled *experiment runners* for parallelization of simulation runs, a central simulation *experiment control center*, and a *time series data store* for the unified provisioning of historic market data as agent bootstrapping for simulations.

The agile market engineering process model as well as the aforementioned software artifacts were successfully used to develop different market platforms. All of them haven been fully operational and were open to the public. In particular, several successive prediction market platforms were built, all of them based on the initial continuous double auction market template that was developed as part of this thesis: *EM-Stoxx* was a sports market where stocks of soccer teams participating in European Soccer Cup 2008 were traded. Based on the insights of this initial prediction market platform, the *Australian Knowledge Exchange (AKX)* prediction market was developed as successor of EM-Stoxx and operated for almost one year under the lead of Stathel et al. (2009). This market was used to forecast water dam levels in Australia. One year later, the EIX prediction market was developed under the lead of Teschner et al. (2010), as successor to AKX. The EIX market is used to predict economic key performance indicators (e.g. GDP, rate of unemployment, or export surplus) for Germany. As of writing of this thesis the market is open to the public, accessible at `http://www.eix-market.de`, and serving about 950 registered participants. All of the aforementioned projects used the agile market engineering process model for project management to deliver fully functional and stable running market platforms.

Additionally, a proof-of-concept energy market platform for combinatorial heat and power trading was developed based on the call market template using the agile market engineering process model for project management. The market was demonstrated at the official launch of the E-Energy project MEREGIO and published under open source license at `http://www.microgridmarket.sourceforge.net`. As of May 2010 it has been downloaded more than 400 times.

In November 2009 another project, the Trading Agent Competition for the Energy Market (TAC Energy) has been started. The objective of this project is to design an open, competitive market simulation platform that will be used to produce robust research results on the structure and operation of retail power markets as well as on automating market interaction by means of competitively tested and benchmarked agents (Block et al., 2009, 2010a). Also the development of this platform has been guided by the agile market engineering process model. Technically, the platform builds on top of the cda market template and uses several components of the experiment center's agent based simulation environment. A first version of the competition has been released and is open to the public at `http://tacenergy.org`. A first international TAC Energy tournament is planned for 2011 (Block et al., 2010b).

6.2 Critical Assessment

Although the agile market engineering process model and its supporting software artifacts have been successfully used throughout several market engineering projects, several limitations pertaining to the process model, to the software artifacts, and to the evaluation method have to be noted.

First, the agile market engineering process model has not yet been applied in projects that were conducted completely independent from the author of this work. Thus, the accomplishment of the already conducted projects described in Chapter 5 cannot unanimously be traced back to the utilization of the agile market engineering process model but might also depend on the direct or indirect involvement of the author in all of the projects.

Second, the success of the agile market engineering process model itself has not objectively been investigated. Only indirect evidence, like *running market platforms* and *fulfilled project requirements* (see Chapter 5), are used to conclude that the agile market engineering process model is well suited for the development of electronic market platforms. Additionally, the sample of already accomplished projects is small and the type of developed markets is similar to each other. Thus, one cannot conclude that the proposed process model and the proposed supporting software artifacts are also well suited for the development of different types of electronic markets. Last but not least all of the projects were small in terms of project team size and in terms of overall budget. It is thus not guaranteed, that the process model is also applicable to large-scale projects with big project teams and large project budgets "at risk" though this limitation is explicitly stated in the beginning of Chapter 4.

Third, concerning the Market Design Knowledge Base (MDKB) an incentive problem has been identified that leads to little or no effort for market development teams to provide their knowledge to the MDKB. The problem is that those persons who donate their knowledge are usually not the same persons who profit from it. More severely, the donator's knowledge loses its uniqueness and thus might weaken his position in his company. Thus he has no incentive to report the knowledge to the MDBK, which leads to little or no knowledge being stored at all.

Fourth, a downside of the current approach of providing specific market templates as foundations for new market platforms to build on, becomes apparent as soon as adaptations or extensions are applied to the original market templates. Currently, after each such change to a template's code base a market developer manually needs to determine if these changes also need to be applied to spawned child market instances too. This process is not trivial as child markets originally inherited the code base from the corresponding market template but might have changed it in the mean time. As a result, this manual update process is cumbersome and error prone and clearly needs to be improved in future.

6.3 Future Work

The agile market engineering process and its supporting software artifacts have been successfully used to develop and run several real-life electronic market platforms. Still several areas for further improvement remain. These are described in more detail in the following subsections.

Additional Market Templates

As of today, the market repository only contains double auction market templates of different flavors and a specific prediction market template. The reason for this is that all agile market engineering projects conducted so far required one of these three market mechanisms to build their market platforms upon. In future this will change. E.g. within the TAC Energy project, a negotiation platform needs to be developed that support semi or fully automated tariff negotiations in an energy market context. After its development the negotiation mechanism can be added to the market repository for future contract negotiation platforms to build on.

Better Maintenance of Market Templates

One potential approach to overcome the template maintenance problem described in Section 6.2 might be to provide the complete initial market functionality in the form of a plugin, which can be installed into a blank project. Such a plugin would provide exactly the same default market functionality the corresponding market template offers right now. The difference is that the plugins code base is located in a different directory than the project's own code base. Only during runtime the plugin and the project code base are then compiled together with the project's own codebase dynamically overwriting plugin functionality in the compiled version of the market platform. Like this market developers can change market functionality by simple writing their own implementations of, say, a market information service. But as the plugin's source code and the project's source code are kept in strictly separated locations, plugin updates can be conducted as soon as they become available. During the next compilation of the market platform the (possibly changed) default market functionality of the plugin is then overridden again by the specific functionality from the project's codebase. Likewise, a customized market functionality that is no longer required can be simply reset to the default behavior by deleting the corresponding source code from the project's code base and by subsequently recompiling the market.

Still, in cases where the fundamental market design is revised (e.g. the market information service is completely removed) updating the plugin may break the market's code integrity. Still for (more likely and more frequent) small changes to the plugin's code base this plugin concept will likely improve overall usability of the market repository and its default templates.

Standardized Decision Support Extensions for Market Templates

Currently the default market instances provide only limited decision support, e.g. by allowing simple search and sorting functionalities for product lookup and selection. Still several specific decision support functionalities have been realized within several of the previously described market projects (see e.g Section 5.3 and 5.4). Clearly, much space for improvement is left. For example, data mining functionality for price and feature comparison could be developed to support market participants in their search efforts. Similarly, specialized market monitoring and analysis tools could be developed that facilitate market monitoring beyond the tools presented in Section 4.1.3 or the ones implemented in the EIX market (see Section 5.4). With an increasing number of DSS extensions their handling should be standardized as well. Also in this case a plugin approach similar to the one described above could be used to manage and easily install these add-on market functionalities on demand if this is required for a particular market project.

Trading Agent Templates

Besides standard market mechanisms the market repository could be extended to also store and provide standard trading agents (Gjerstad and Dickhaut, 2001, see e.g.) as templates for new electronic agent development projects to build upon. Automated trading becomes more and more important in electronic commerce representing more than 50% of the overall trading volume in some financial markets (Hendershott and Riordan, 2009).

For example, the trading agents developed during future TAC Energy competitions could be collected and stored in a central repository for new teams to build their own trading agents upon. A first demo agent for the competition is currently developed and its source code is made available under open source license at [1]. Similar agents could be developed and centrally managed for all different types of markets that are provided by the market repository.

Payment-per-Service

As of today the default market templates ship with a specific *billing service* component that provides default functionality to for charging market participants sign-up or transaction fees. In future each particular market service (e.g. clearing service, pricing service, market information service, etc. – see Section 4.2.2) could be equipped with a dedicated billing component on its own. A market participant could then be charged more fine grained fees, such as, for example, a transaction fee for each request to the information service. This type of billing would also allow for a simple decoupling and outsourcing of the different market services. By default, all of them are currently operated by one single market operator. With a per-service billing in place, the market services could be outsourced to specialized service providers each of them charging a separate fee for its service. On the technical level this could increases scalability of the market platform as each of the services could be operated separately on independent hardware systems. On the business level, a per-service billing approach would provide the foundation for a legal separation of the service providers. Service separation and outsourcing is a common practice in several industries. For example, trading and settlement services in the financial industry are traditionally provided by distinct legal entities.

[1] `https://launchpad.net/tacenergydemo`

In summary, the agile market engineering process model and the supporting software artifacts developed throughout this thesis project have been successfully applied in several real-life market engineering projects aimed at developing and operating publicly accessible electronic market platforms. The process model as well as its accompanying software tools represent *"purposeful, novel, and innovative artifacts"* that target a *"relevant problem"* and *"utilize available means to reach desired ends while satisfying laws in the problem domain."* (Hevner et al., 2004). This thesis itself as well as several previously published articles on certain aspects of this work contribute to the communication of the results.

A

Notation

symbol	explanation	units
$\mathcal{B}$	The set of brokers	
$\mathcal{C}$	Set of energy consumers	
$\mathcal{G}$	Set of energy producers (generators)	
t	A particular time, typically the current simulation time (now)	time point
τ	timeslot duration	minutes
$\mathcal{S}$	set of all timeslots	
s_n	the current timeslot	time interval
t_n	start time of timeslot s_n	time point
e	energy (typically, power integrated over a timeslot)	kilowatt-hours (kWh)
e_c	load (energy consumed)	kWh
$e_c(i,s)$	(maximum) load of consumer i in timeslot s	kWh
$\epsilon_c(i,s)$	controllable load (balancing capacity) for consumer i in timeslot s	kWh
$\epsilon_c(b,s)$	controllable load (load portion of balancing capacity) for broker b in timeslot s	kWh
e_g	production (energy generated)	kWh
$e_g(j,s)$	(maximum) generation capacity of producer j in timeslot s	kWh
$\epsilon_g(j,s)$	controllable production (generation portion of balancing power capacity) for producer j in timeslot s	kWh
$\epsilon_g(b,s)$	controllable portion of total production capacity for broker b in timeslot s	kWh
e'_c	forecasted load (energy consumption)	kWh
$e'_c(i,s)$	forecasted load of consumer i in timeslot s	kWh
$e'_c(b,s)$	overall forecasted load of broker b in timeslot s	kWh
e'_g	forecasted production capacity (energy production)	kWh
$e'_g(j,s)$	forecasted energy production of producer j in timeslot s	kWh
$e'_g(b,s)$	overall forecasted energy production of broker b in timeslot s	kWh

symbol	**explanation**	**units**
$\zeta_g(j, s)$	Forecasting uncertainty for energy output of generator j in timeslot s	kWh
p, p'	prices	\$
$p_{j,s}$	price for production, generator j, timeslot s	\$/kWh
$p_{i,s}$	price for consumption, consumer i, timeslot s	\$/kWh
$PE_g(j)$	Price elasticity for generator j	kWh^2/\$
$\hat{e}_g(j, s, p)$	Forecast for generator j in timeslot s, given production price p	kWh
$PE_c(i)$	Price elasticity for consumer i	kWh^2/\$
$\hat{e}_c(i, s, p)$	Forecast for consumer c in timeslot s, given consumption price p	kWh

B

AuctionBot & meet2trade Configuration

AuctionBot Configuration

Listed below is a sample AuctionBot configuration that can be used to instantiate and run a specific auction on the AuctionBot platform (Taylor and Wurman, 2000).

```

<auction>

  <roundInformation>
    <currentRound>1</currentRound>
    <numberOfRounds>12</numberOfRounds>
  </roundInformation>

  <biddingRules>
    <multipleBuyers>true</multipleBuyers>
    <multipleSellers>false</multipleSellers>
    <bidDivisible>false</bidDivisible>
    <activity>n/a</activity>
    <sellerBidDominance>none</sellerBidDominance>
    <buyerBidDominance>none</buyerBidDominance>
    <expressiveness>singleUnit</expressiveness>
    <beatTheQuote>
      <agent>buyer</agent>
    </beatTheQuote>
    <withdrawelAllowed>false</withdrawelAllowed>
  </biddingRules>

  <informationRevelation>
    <quoteBid>false</quoteBid>
    <quoteAsk>true</quoteAsk>
    <quoteTiming><activity/></quoteTiming>
    <orderBook>winner</orderBook>
    <showPrices>false</showPrices>
    <showQuantity>true</showQuantity>
    <showIdentity>true</showIdentity>
  </informationRevelation>

  <clearingPolicy>
    <clearTiming><scheduled/></clearTiming>
    <closingCondition><scheduled/></closingCondition>
    <matchingFunction>k=1</matchingFunction>
    <tieBreaker>earliest</tieBreaker>
    <auctioneerFees>entrance</auctioneerFees>
    <auctioneerFees>nonlinear</auctioneerFees>
  </clearingPolicy>

</auction>
```

Sample meet2trade MML instance

Listed below is a sample meet2trade configuration that can be used to instantiate and run double auction on the meet2trade platform. In particular this configuration mimics the behavior of large stock exchanges that usually open a trading day with a call auction and then switch to continuous trading through a continuous double auction mechanism (Weinhardt et al., 2006).

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<MML>
    <MarketModel>
        <MarketModelName>Call auction</MarketModelName>
        <ProductCategoryID>1</ProductCategoryID>
        <AuctionType>CallMarket</AuctionType>
        <VisibleInformationFromOrderbook>
            <Buy>
                <VisibleDepth>0</VisibleDepth>
                <AccumulationProperties>no accumulation</
                     AccumulationProperties>
                <propertyKeys>
                    <key>volume</key>
                    <pos>1</pos>
                </propertyKeys>
                <propertyKeys>
                    <key>price</key>
                    <pos>2</pos>
                </propertyKeys>
            </Buy>
            <Sell>
                <VisibleDepth>0</VisibleDepth>
                <AccumulationProperties>no accumulation</
                     AccumulationProperties>
                <propertyKeys>
                    <key>volume</key>
                    <pos>1</pos>
                </propertyKeys>
                <propertyKeys>
                    <key>price</key>
                    <pos>2</pos>
                </propertyKeys>
            </Sell>
        </VisibleInformationFromOrderbook>
        <OfferLimitation/>
        <WithDrawBid>false</WithDrawBid>
        <PriceFunction>
            <FirstOrder>true</FirstOrder>
        </PriceFunction>
        <Description>The call auction is used to "get the market started".</
             Description>
        <PartialExecution>false</PartialExecution>
        <MatchExecutionRules>
            <MatchingExecution durationTime="60000" startingTime="300000"/>
            <LiquidityTriggeredMatchExecution periodOfLiquidityProofing="-1"
                  startTimeForLiquidityProofing="-1" liquidityBuySide="-1"
                 liquiditySellSide="-1"/>
            <TimeExtendedMatchExecution numberOfExtensions="-1"
                 extensionForMatching="-1"/>
        </MatchExecutionRules>
        <MultiAttributeTrading/>
    </MarketModel>
    <MarketModel>
        <MarketModelName>CDA </MarketModelName>
        <ProductCategoryID>1</ProductCategoryID>
        <AuctionType>CDA</AuctionType>
        <AllowedOrderTypes>discretionary</AllowedOrderTypes>
        <AllowedOrderTypes>relative</AllowedOrderTypes>
        <AllowedOrderTypes>bracket</AllowedOrderTypes>
        <AllowedOrderTypes>trailing</AllowedOrderTypes>
        <AllowedOrderTypes>stop</AllowedOrderTypes>
        <AllowedOrderTypes>unlimit</AllowedOrderTypes>
        <AllowedOrderTypes>limit</AllowedOrderTypes>
        <VisibleInformationFromOrderbook>
            <Buy>
                <VisibleDepth>0</VisibleDepth>
                <AccumulationProperties>no accumulation</
                    AccumulationProperties>
                <propertyKeys>
                    <key>volume</key>
                    <pos>1</pos>
                </propertyKeys>
                <propertyKeys>
                    <key>price</key>
                    <pos>2</pos>
                </propertyKeys>
            </Buy>
            <Sell>
                <VisibleDepth>0</VisibleDepth>
                <AccumulationProperties>no accumulation</
                    AccumulationProperties>
                <propertyKeys>
                    <key>volume</key>
                    <pos>1</pos>
                </propertyKeys>
                <propertyKeys>
                    <key>price</key>
                    <pos>2</pos>
                </propertyKeys>
            </Sell>
        </VisibleInformationFromOrderbook>
        <OfferLimitation/>
        <WithDrawBid>false</WithDrawBid>
        <PriceFunction>
            <FirstOrder>true</FirstOrder>
        </PriceFunction>
        <Description>CDA for the rest of the meta market runtime</
            Description>
        <PartialExecution>false</PartialExecution>
        <MatchExecutionRules>
            <MatchingExecution/>
            <LiquidityTriggeredMatchExecution periodOfLiquidityProofing="-1"
                  startTimeForLiquidityProofing="-1" liquidityBuySide="0"
                liquiditySellSide="0"/>
            <TimeExtendedMatchExecution/>
        </MatchExecutionRules>
        <MultiAttributeTrading isMultiAttributetrading="false"/>
    </MarketModel>
    <MetaMarket>
        <MarketAttributes>
            <MarketStartingRule>
                <TimeRule>
                    <Time>1000</Time>
                </TimeRule>
            </MarketStartingRule>
            <MarketStoppingRule>
                <TimeRule>
                    <Time>300000</Time>
                </TimeRule>
            </MarketStoppingRule>
            <MarketName>Call auction</MarketName>
        </MarketAttributes>
        <MarketAttributes>
            <MarketStartingRule>
                <EventRule>
                    <MarketGestoppt>
                        <MarketNameStopped>Call auction</MarketNameStopped>
                    </MarketGestoppt>
                </EventRule>
            </MarketStartingRule>
            <MarketStoppingRule>
                <TimeRule>
```

```
                    <Time>1800000</Time>
                </TimeRule>
                <EventRule>
                    <Volatility>
                        <RelativAmount>5.0</RelativAmount>
                        <TimeIntervall>60000</TimeIntervall>
                        <AbsolutAmount>-1.0</AbsolutAmount>
                        <NumberOfAllowedInterruptions>5</
                            NumberOfAllowedInterruptions>
                    </Volatility>
                </EventRule>
            </MarketStoppingRule>
            <MarketName>CDA </MarketName>
        </MarketAttributes>
        <MetaMarketAttributes>
            <Name>Xetra Meta Market</Name>
            <ProductCategoryId>1</ProductCategoryId>
            <ProductCategoryDescription>Stock</ProductCategoryDescription>
            <MetaMarketInfo>This is a demo xetra market with Call auction
                and CDA</MetaMarketInfo>
            <Title>title</Title>
            <NumberOfOrdersPerUser>100</NumberOfOrdersPerUser>
            <ProductInformation/>
            <AuctionInfoID>1</AuctionInfoID>
            <UserIdVisible>true</UserIdVisible>
            <AllUsersAllowed>true</AllUsersAllowed>
            <OrderIdVisible>true</OrderIdVisible>
            <OrderBookVisible>true</OrderBookVisible>
            <TimeStampVisible>true</TimeStampVisible>
            <UserId>1</UserId>
            <UserNameVisible>true</UserNameVisible>
            <UserName>jma</UserName>
            <AllowedProducts>
                <ProductID>1</ProductID>
                <ProductIdDescription>BMW</ProductIdDescription>
            </AllowedProducts>
            <StartRule>
                <Time>1164631320000</Time>
            </StartRule>
            <StopRule>
                <Time>1164633139430</Time>
            </StopRule>
        </MetaMarketAttributes>
    </MetaMarket>
</MML>
```

C

Definition of Done

The following *Definition of Done* was developed and used in the TAC Energy project:

Definition of Done

A user story is done if all of its identified tasks are:

- implemented
- reviewed
- tested
- documented
- deployed
- making the product owner happy

The remainder of this document details the parts of "done":

Implemented: Implemented code adheres to the following principles:

- adheres to coding style guide[1]
- code is compiled and runs against the current "HEAD" version in source control
- code is commented according to the coding style guide
- all FIXME/TODO tags in the code have been addressed. If they are not resolved (yet) the product owner has to explicitly agree that thy story still is "done"

It must also be possible to automatically build the implemented code. This will be accomplished by Hudson CI server, which automatically creates a binary release of the latest HEAD version from the code repository as soon as a code change is checked in.

[1] `http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html`

Reviewed: Every task must be peer reviewed. There is no formal process how this should be done. Just make sure that a colleague has a look at the changes you made an incorporate his / her feedback.

Tested: Make sure that all changes are thoroughly tested to keep overall code quality consistently high. For Java / Groovy Code this means:

- Write unit tests for all non-supertrivial code
- Make sure that Cobertura reports a solid code coverage for your piece of code (`grails test-app -coverage` runs Cobertura automatically for you on your local machine and hudson will do so on the CI server as well)
- Write integration tests to automatically test the end-to-end scenario described in the user story (involves creating / manipulating database objects, like e.g. orders or user depots)
- Write Selenium based tests if there is a web interface involved

Deployed: Code must be deployed on the production machine (currently `http://ibwmarkets.iw.uni-karlsruhe.de`)

Documented: The documentation must allow a reasonably knowledgeable person to understand the implemented code. To check this level of documentation, reviews should be usually handed over just pointing the reviewer to the code documentation (java / groovy doc) and / or the documentation generated through the grails documentation engine (check `http://www.tacenergy.org/docs/latest/manual/index.html` to see how generated documentation of the grails documentation engine looks like). In addition you have to clearly state licensing conditions whenever you use a 3rd party API or tool if that license is different from our currently used one (for TAC Energy this is Apache 2 License)

Making the Product Owner Happy: Ask the product owner on how satisfied he is with the implementation of the user story (best before the review meeting). Please note that we are not developing for ourselves or for the product owner but for the prospect students that have to use the framework during the summer term as the basis for achieving good grades!

D

Order Processing in Call and CDA Market Templates

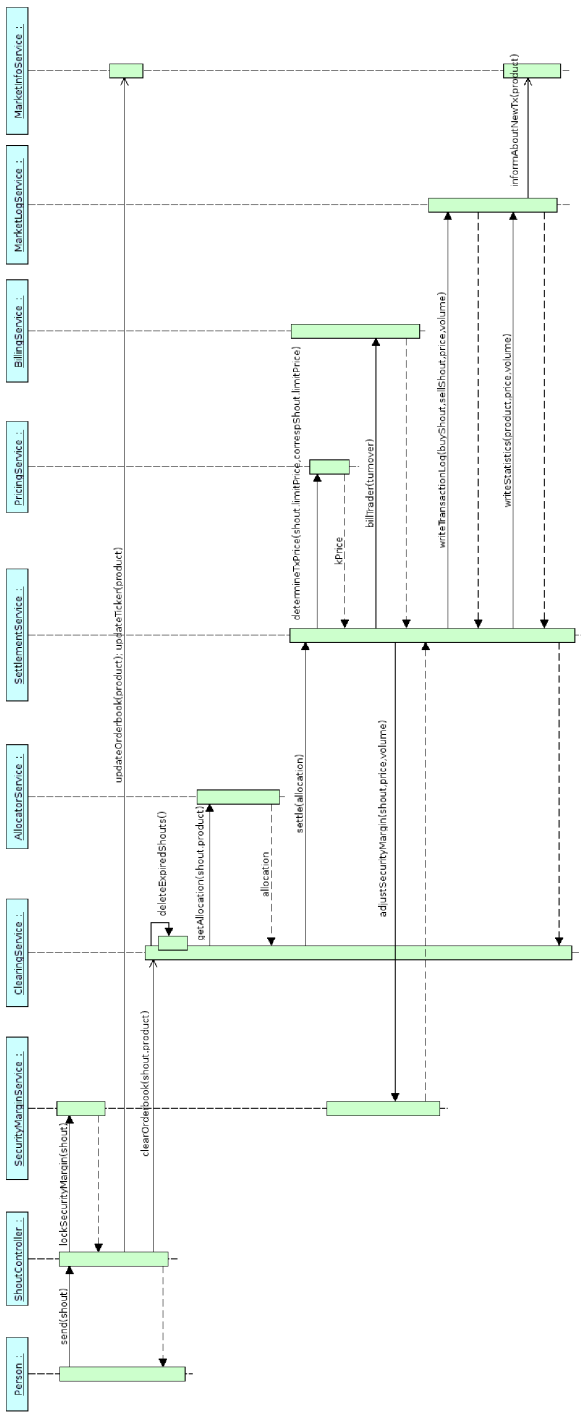

Person :
ShoutController :
SecurityMarginService :
ClearingService :
AllocatorService :
SettlementService :
PricingService :
BillingService :
MarketLogService :
MarketInfoService :
send(shout)
lockSecurityMargin(shout)
clearOrderbook(shout,product)
updateOrderbook(product); updateTicker(product)
deleteExpiredShouts()
getAllocation(shout,product)
allocation
settle(allocation)
determineTxPrice(shout.limitPrice,correspShout.limitPrice)
kPrice
billTrader(turnover)
adjustSecurityMargin(shout,price,volume)
writeTransactionLog(buyShout,sellShout,price,volume)
writeStatistics(product,price,volume)
informAboutNewTx(product)

Fig. D.1. Order Process in CDA Market Template

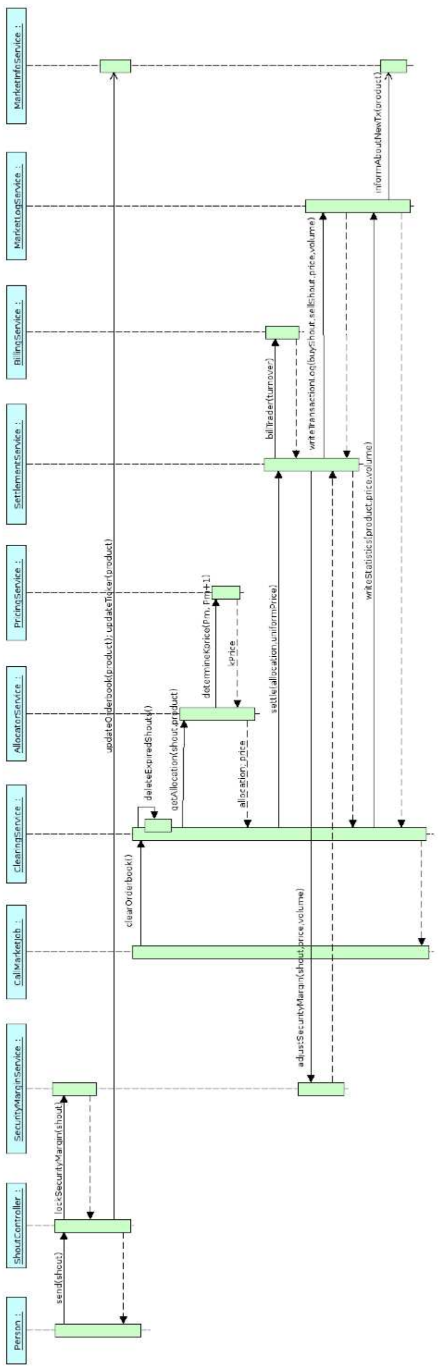
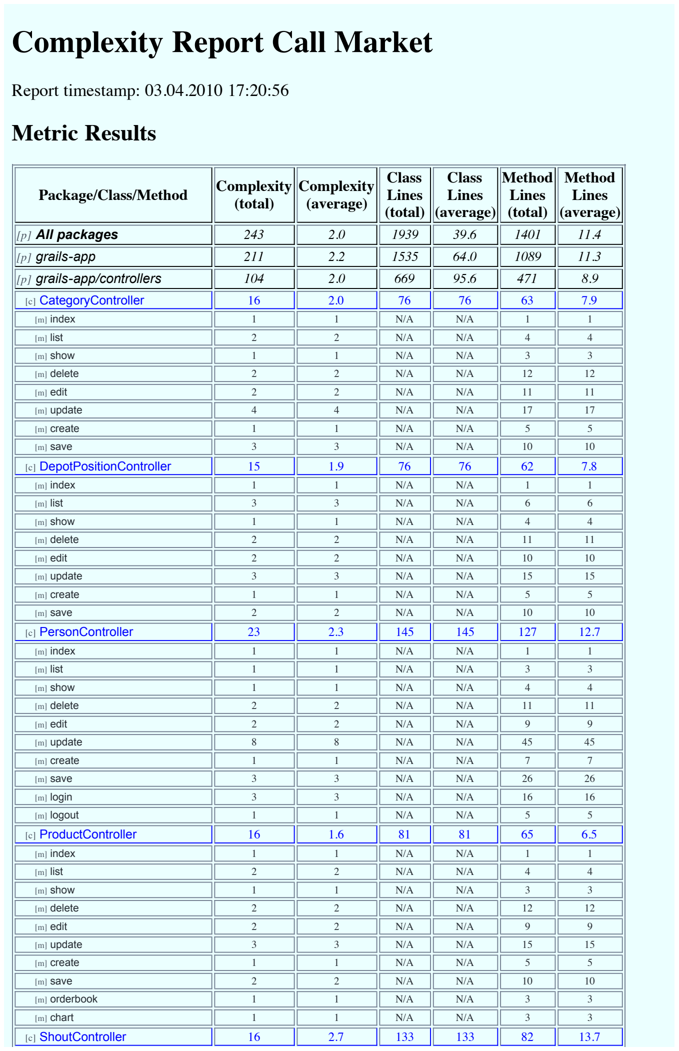

Person :
ShoutController :
SecurityMarginService :
CallMarketJob :
ClearingService :
AllocatorService :
PricingService :
SettlementService :
BillingService :
MarketLogService :
MarketInfoService :
send(shout)
lockSecurityMargin(shout)
updateOrderbook(product); updateTicker(product)
clearOrderbook()
deleteExpiredShouts()
getAllocation(shout,product)
determineKprice(Pm, Pm+1)
kPrice
allocation, price
settle(allocation,uniformPrice)
adjustSecurityMargin(shout,price,volume)
billTrader(turnover)
writeTransactionLog(buyShout,sellShout,price,volume)
writeStatistics(product,price,volume)
informAboutNewTx(product)

Fig. D.2. Order Processing in Call Market Template

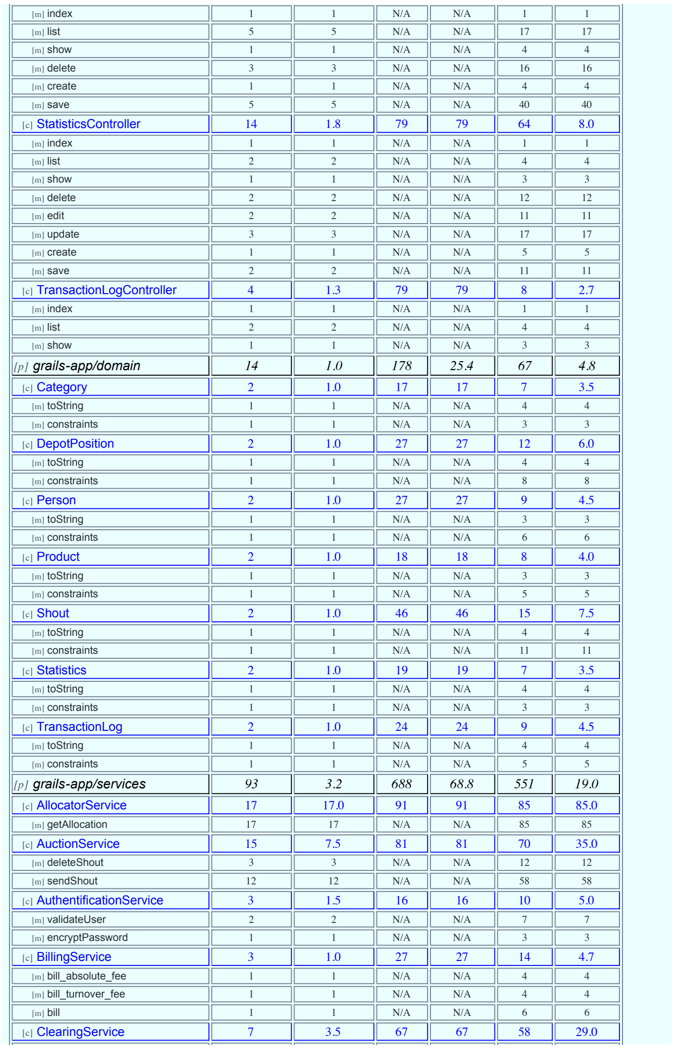
E Complexity Reports for Market Templates



Complexity Report Call Market

Report timestamp: 03.04.2010 17:20:56

Metric Results



Package/Class/Method	Complexity (total)	Complexity (average)	Class Lines (total)	Class Lines (average)	Method Lines (total)	Method Lines (average)
[p] ***All packages***	*243*	*2.0*	*1939*	*39.6*	*1401*	*11.4*
[p] *grails-app*	*211*	*2.2*	*1535*	*64.0*	*1089*	*11.3*
[p] *grails-app/controllers*	*104*	*2.0*	*669*	*95.6*	*471*	*8.9*
[c] CategoryController	16	2.0	76	76	63	7.9
[m] index	1	1	N/A	N/A	1	1
[m] list	2	2	N/A	N/A	4	4
[m] show	1	1	N/A	N/A	3	3
[m] delete	2	2	N/A	N/A	12	12
[m] edit	2	2	N/A	N/A	11	11
[m] update	4	4	N/A	N/A	17	17
[m] create	1	1	N/A	N/A	5	5
[m] save	3	3	N/A	N/A	10	10
[c] DepotPositionController	15	1.9	76	76	62	7.8
[m] index	1	1	N/A	N/A	1	1
[m] list	3	3	N/A	N/A	6	6
[m] show	1	1	N/A	N/A	4	4
[m] delete	2	2	N/A	N/A	11	11
[m] edit	2	2	N/A	N/A	10	10
[m] update	3	3	N/A	N/A	15	15
[m] create	1	1	N/A	N/A	5	5
[m] save	2	2	N/A	N/A	10	10
[c] PersonController	23	2.3	145	145	127	12.7
[m] index	1	1	N/A	N/A	1	1
[m] list	1	1	N/A	N/A	3	3
[m] show	1	1	N/A	N/A	4	4
[m] delete	2	2	N/A	N/A	11	11
[m] edit	2	2	N/A	N/A	9	9
[m] update	8	8	N/A	N/A	45	45
[m] create	1	1	N/A	N/A	7	7
[m] save	3	3	N/A	N/A	26	26
[m] login	3	3	N/A	N/A	16	16
[m] logout	1	1	N/A	N/A	5	5
[c] ProductController	16	1.6	81	81	65	6.5
[m] index	1	1	N/A	N/A	1	1
[m] list	2	2	N/A	N/A	4	4
[m] show	1	1	N/A	N/A	3	3
[m] delete	2	2	N/A	N/A	12	12
[m] edit	2	2	N/A	N/A	9	9
[m] update	3	3	N/A	N/A	15	15
[m] create	1	1	N/A	N/A	5	5
[m] save	2	2	N/A	N/A	10	10
[m] orderbook	1	1	N/A	N/A	3	3
[m] chart	1	1	N/A	N/A	3	3
[c] ShoutController	16	2.7	133	133	82	13.7
[m] index	1	1	N/A	N/A	1	1
[m] list	5	5	N/A	N/A	17	17
[m] show	1	1	N/A	N/A	4	4
[m] delete	3	3	N/A	N/A	16	16
[m] create	1	1	N/A	N/A	4	4
[m] save	5	5	N/A	N/A	40	40
[c] StatisticsController	14	1.8	79	79	64	8.0
[m] index	1	1	N/A	N/A	1	1
[m] list	2	2	N/A	N/A	4	4
[m] show	1	1	N/A	N/A	3	3
[m] delete	2	2	N/A	N/A	12	12
[m] edit	2	2	N/A	N/A	11	11
[m] update	3	3	N/A	N/A	17	17
[m] create	1	1	N/A	N/A	5	5
[m] save	2	2	N/A	N/A	11	11
[c] TransactionLogController	4	1.3	79	79	8	2.7
[m] index	1	1	N/A	N/A	1	1
[m] list	2	2	N/A	N/A	4	4
[m] show	1	1	N/A	N/A	3	3
[p] *grails-app/domain*	*14*	*1.0*	*178*	*25.4*	*67*	*4.8*
[c] Category	2	1.0	17	17	7	3.5
[m] toString	1	1	N/A	N/A	4	4
[m] constraints	1	1	N/A	N/A	3	3
[c] DepotPosition	2	1.0	27	27	12	6.0
[m] toString	1	1	N/A	N/A	4	4
[m] constraints	1	1	N/A	N/A	8	8
[c] Person	2	1.0	27	27	9	4.5
[m] toString	1	1	N/A	N/A	3	3
[m] constraints	1	1	N/A	N/A	6	6
[c] Product	2	1.0	18	18	8	4.0
[m] toString	1	1	N/A	N/A	3	3
[m] constraints	1	1	N/A	N/A	5	5
[c] Shout	2	1.0	46	46	15	7.5
[m] toString	1	1	N/A	N/A	4	4
[m] constraints	1	1	N/A	N/A	11	11
[c] Statistics	2	1.0	19	19	7	3.5
[m] toString	1	1	N/A	N/A	4	4
[m] constraints	1	1	N/A	N/A	3	3
[c] TransactionLog	2	1.0	24	24	9	4.5
[m] toString	1	1	N/A	N/A	4	4
[m] constraints	1	1	N/A	N/A	5	5
[p] *grails-app/services*	*93*	*3.2*	*688*	*68.8*	*551*	*19.0*
[c] AllocatorService	17	17.0	91	91	85	85.0
[m] getAllocation	17	17	N/A	N/A	85	85
[c] AuctionService	15	7.5	81	81	70	35.0
[m] deleteShout	3	3	N/A	N/A	12	12
[m] sendShout	12	12	N/A	N/A	58	58
[c] AuthentificationService	3	1.5	16	16	10	5.0
[m] validateUser	2	2	N/A	N/A	7	7
[m] encryptPassword	1	1	N/A	N/A	3	3
[c] BillingService	3	1.0	27	27	14	4.7
[m] bill_absolute_fee	1	1	N/A	N/A	4	4
[m] bill_turnover_fee	1	1	N/A	N/A	4	4
[m] bill	1	1	N/A	N/A	6	6
[c] ClearingService	7	3.5	67	67	58	29.0

Complexity Report CDA Market

Report timestamp: 03.04.2010 17:17:25

Metric Results

Package/Class/Method	Complexity (total)	Complexity (average)	Class Lines (total)	Class Lines (average)	Method Lines (total)	Method Lines (average)
[p] ***All packages***	*243*	*2.0*	*1942*	*39.6*	*1399*	*11.4*
[p] *grails-app*	*211*	*2.2*	*1538*	*64.1*	*1087*	*11.3*
[p] *grails-app/controllers*	*104*	*2.0*	*668*	*95.4*	*470*	*8.9*
[c] CategoryController	16	2.0	76	76	63	7.9
[m] index	1	1	N/A	N/A	1	1
[m] list	2	2	N/A	N/A	4	4
[m] show	1	1	N/A	N/A	3	3
[m] delete	2	2	N/A	N/A	12	12
[m] edit	2	2	N/A	N/A	11	11
[m] update	4	4	N/A	N/A	17	17
[m] create	1	1	N/A	N/A	5	5
[m] save	3	3	N/A	N/A	10	10
[c] DepotPositionController	15	1.9	76	76	62	7.8
[m] index	1	1	N/A	N/A	1	1
[m] list	3	3	N/A	N/A	6	6
[m] show	1	1	N/A	N/A	4	4
[m] delete	2	2	N/A	N/A	11	11
[m] edit	2	2	N/A	N/A	10	10
[m] update	3	3	N/A	N/A	15	15
[m] create	1	1	N/A	N/A	5	5
[m] save	2	2	N/A	N/A	10	10
[c] PersonController	23	2.3	145	145	127	12.7
[m] index	1	1	N/A	N/A	1	1
[m] list	1	1	N/A	N/A	3	3
[m] show	1	1	N/A	N/A	4	4
[m] delete	2	2	N/A	N/A	11	11
[m] edit	2	2	N/A	N/A	9	9
[m] update	8	8	N/A	N/A	45	45
[m] create	1	1	N/A	N/A	7	7
[m] save	3	3	N/A	N/A	26	26
[m] login	3	3	N/A	N/A	16	16
[m] logout	1	1	N/A	N/A	5	5
[c] ProductController	16	1.6	81	81	65	6.5
[m] index	1	1	N/A	N/A	1	1
[m] list	2	2	N/A	N/A	4	4
[m] show	1	1	N/A	N/A	3	3
[m] delete	2	2	N/A	N/A	12	12
[m] edit	2	2	N/A	N/A	9	9
[m] update	3	3	N/A	N/A	15	15
[m] create	1	1	N/A	N/A	5	5
[m] save	2	2	N/A	N/A	10	10
[m] orderbook	1	1	N/A	N/A	3	3
[m] chart	1	1	N/A	N/A	3	3
[c] ShoutController	16	2.7	132	132	81	13.5
[m] index	1	1	N/A	N/A	1	1
[m] list	5	5	N/A	N/A	17	17
[m] show	1	1	N/A	N/A	4	4
[m] delete	3	3	N/A	N/A	16	16
[m] create	1	1	N/A	N/A	4	4
[m] save	5	5	N/A	N/A	39	39
[c] StatisticsController	14	1.8	79	79	64	8.0
[m] index	1	1	N/A	N/A	1	1
[m] list	2	2	N/A	N/A	4	4
[m] show	1	1	N/A	N/A	3	3
[m] delete	2	2	N/A	N/A	12	12
[m] edit	2	2	N/A	N/A	11	11
[m] update	3	3	N/A	N/A	17	17
[m] create	1	1	N/A	N/A	5	5
[m] save	2	2	N/A	N/A	11	11
[c] TransactionLogController	4	1.3	79	79	8	2.7
[m] index	1	1	N/A	N/A	1	1
[m] list	2	2	N/A	N/A	4	4
[m] show	1	1	N/A	N/A	3	3
[p] *grails-app/domain*	*14*	*1.0*	*179*	*25.6*	*67*	*4.8*
[c] Category	2	1.0	17	17	7	3.5
[m] toString	1	1	N/A	N/A	4	4
[m] constraints	1	1	N/A	N/A	3	3
[c] DepotPosition	2	1.0	27	27	12	6.0
[m] toString	1	1	N/A	N/A	4	4
[m] constraints	1	1	N/A	N/A	8	8
[c] Person	2	1.0	28	28	9	4.5
[m] toString	1	1	N/A	N/A	3	3
[m] constraints	1	1	N/A	N/A	6	6
[c] Product	2	1.0	18	18	8	4.0
[m] toString	1	1	N/A	N/A	3	3
[m] constraints	1	1	N/A	N/A	5	5
[c] Shout	2	1.0	46	46	15	7.5
[m] toString	1	1	N/A	N/A	4	4
[m] constraints	1	1	N/A	N/A	11	11
[c] Statistics	2	1.0	19	19	7	3.5
[m] toString	1	1	N/A	N/A	4	4
[m] constraints	1	1	N/A	N/A	3	3
[c] TransactionLog	2	1.0	24	24	9	4.5
[m] toString	1	1	N/A	N/A	4	4
[m] constraints	1	1	N/A	N/A	5	5
[p] *grails-app/services*	*93*	*3.2*	*691*	*69.1*	*550*	*19.0*
[c] AllocatorService	17	17.0	91	91	85	85.0
[m] getAllocation	17	17	N/A	N/A	85	85
[c] AuctionService	15	7.5	81	81	70	35.0
[m] deleteShout	3	3	N/A	N/A	12	12
[m] sendShout	12	12	N/A	N/A	58	58
[c] AuthentificationService	3	1.5	16	16	10	5.0
[m] validateUser	2	2	N/A	N/A	7	7
[m] encryptPassword	1	1	N/A	N/A	3	3
[c] BillingService	3	1.0	27	27	14	4.7
[m] bill_absolute_fee	1	1	N/A	N/A	4	4
[m] bill_turnover_fee	1	1	N/A	N/A	4	4
[m] bill	1	1	N/A	N/A	6	6
[c] ClearingService	7	3.5	68	68	56	28.0

References

Abrahamsson, P., K. Conboy, and X. Wang (2009, August). 'Lots done, more to do': the current state of agile systems development research. *European Journal of Information Systems 18*(4), 281–284.

Adam, M. T., M. Hagenau, D. Neumann, and C. Weinhardt (2008). Emotions in Electronic Auctions - A Physio-Economic Approach on Information Systems. In *Proceedings of the 16th European Conference on Information Systems (ECIS' 08)*, Galway, Ireland, pp. 315–325.

Ahlert, K.-H. and C. Block (2010). Assessing the impact of price forecast errors on the economics of distributed storage systems. In *43rd Hawaii International Conference on System Science (HICSS-43)*, Hawaii, USA. (forthcoming).

Alur, D., D. Malks, and J. Crupi (2003). *Core J2EE Patterns: Best Practices and Design Strategies* (2 ed.). Prentice Hall.

Anandalingam, G., R. W. Day, and S. Raghavan (2005). The landscape of electronic market design. *Management Science 51*(3), 316–327.

Ash, S. (2007, October). Moscow prioritisation. Technical report, DSDM Consortiu,.

Bartolini, C., C. Preist, and N. R. Jennings (2005). A software framework for automated negotiation. In *Software Engineering for Multi-Agent Systems III*, pp. 213–235.

Batory, D. (1994). The leaps algorithm. Technical report, University of Texas at Austin.

Bauer, O. and S. Czajka (2009, December). Online shopping liegt im trend. Technical report, Statistisches Bundesamt.

Beck, K. (1999, Oct). Embracing change with extreme programming. *Computer 32*(10), 70–77.

Beck, K. (2002, November). *Test Driven Development by Example.* Amsterdam, The Netherlands: Addison-Wesley Longman.

Beck, K. and C. Andres (2004). *Extreme programming explained: embrace change.* Addison-Wesley Professional.

Belanoff, P., P. Elbow, and S. I. Fontaine (Eds.) (1991, January). *Nothing Begins with N: New Investigations of Freewriting* (1 ed.). Southern Illinois University Press.

Benisch, M., A. Greenwald, I. Grypari, R. Lederman, V. Naroditskiy, and M. Tschantz (2004). Botticelli: A supply chain management agent designed to optimize under uncertainty. *ACM Trans. on Comp. Logic 4*(3), 29–37.

Bichler, M., C. Beam, and A. Segev (1998). Offer: A broker-centered object framework for electronic requisitioning. In *Trends in Distributed Systems for Electronic Commerce*, pp. 154+.

Blaabjerg, F., R. Teodorescu, M. Liserre, and A. Timbus (2006). Overview of control and grid synchronization for distributed power generation systems. *IEEE Transactions on Industrial Electronics 53*(5), 1398–1409.

Block, C. (2007). Price-based coordination in decentralized micro power grids. In *Group Decision and Negotiation 2007*, Montreal, Canada.

Block, C., F. Bomarius, P. Bretschneider, F. Briegel, N. Burger, B. Fey, H. Frey, J. Hartmann, C. Kern, B. Plail, G. Praehauser, L. Schetters, F. Schoepf, D. Schumann, F. Schwammberger, O. Terzidis, R. Thiemann, C. van Dinther, K. von Sengbusch, A. Weidlich, and C. Weinhardt (2008, 12). Internet of Energy - ICT for Future Energy Markets. Bdi-drucksache nr. 418, Bundesverband der Deutschen Industrie e.V. (BDI).

Block, C. and E. Chen (2007). Fast Prototyping of Electronic Agents for the Web. In G. E. Kersten, J. Rios, and E. Chen (Eds.), *Group Decision and Negotiation (GDN) 2007*, Montreal, Canada.

Block, C., J. Collins, L. Filipova-Neumann, and W. Ketter (2010, June). A competitive testbed for the development of intelligent agents in the energy domain. In *11th Conference on Group Decision and Negotiation (GDN)*, Delft, Netherlands.

Block, C., J. Collins, and W. Ketter (2010a, August). Agent-based competitive simulation: Exploring future retail energy markets. In *Proceedings of the 12th International Conference on Electronic Commerce (ICEC)*, Hawaii, USA.

Block, C., J. Collins, and W. Ketter (2010b, June). Exploring retail energy markets through competitive simulation. In K. Larson (Ed.), *ACM EC 2010 Workshop on Trading Agent Design and Analysis (TADA)*, Harvard University.

Block, C., J. Collins, W. Ketter, and C. Weinhardt (2009). A multi-agent energy trading competition. Technical Report ERS-2009-054-LIS, RSM Erasmus University, Rotterdam, The Netherlands.

Block, C., G. Kersten, H. Gimpel, and C. Weinhardt (2006). Reasons for rejecting Pareto-improvements in negotiations. In S. Seifert and C. Weinhardt (Eds.), *Group Decision and Negotiation (GDN) 2006*, International Conference, Karlsruhe, Germany, June 25-28, 2006.

Block, C. and D. Neumann (2008). A decision support system for choosing market mechanisms in e-procurement. In H. Gimpel, N. R. Jennings, Kersten, G. E., Ockenfels, Axel, and C. Weinhardt (Eds.), *Negotiation, Auctions, and Market Engineering*, Volume 2, pp. 44–57. Springer Berlin Heidelberg.

Block, C., D. Neumann, and C. Weinhardt (2008). A market mechanism for energy allocation in micro-chp grids. In *Hawaii International Conference on System Sciences, Proceedings of the 41st Annual*, pp. 172.

Blume, M., S. Luckner, and C. Weinhardt (2008, December). Fraud detection in play-money prediction markets. *Information Systems and E-Business Management*.

Boehm, B. (1981). Software engineering economics. *Englewood Cliffs*.

Borenstein, S., J. B. Bushnell, and F. A. Wolak (2002). Measuring market inefficiencies in California's restructured wholesale electricity market. *The American Economic Review 92*(5), 1376–1405.

Broy, M. and A. Rausch (2005, June). Das neue v-modell® xt. *Informatik-Spektrum 28*(3), 220–229.

Campbell, C., G. Ray, and W. A. Muhanna (2005). Search and collusion in electronic markets. *Management Science 51*(3), 497–507.

Cao, L., K. Mohan, P. Xu, and B. Ramesh (2009, August). A framework for adapting agile development methodologies. *European Journal of Information Systems 18*(4), 332–343.

Chavez, A. and P. Maes (1996). Kasbah: An agent marketplace for buying and selling goods. In *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, Volume 31, pp. 40.

Chen, E., B. Yu, and K. Kolitz (2006). Negotiation or Auction? The NorA Project. In *Proceedings from Dagstuhl Seminar: Negotiation and Market Engineering.*

Chi, R. H. and M. Y. Kiang (1991). An integrated approach of rule-based and case-based reasoning for decision support. In *CSC '91: Proceedings of the 19th annual conference on Computer Science*, New York, NY, USA, pp. 255–267. ACM Press.

Choros, K. and M. Muskala (2009). Block map technique for the usability evaluation of a website. In N. T. Nguyen, R. Kowalczyk, and S.-M. Chen (Eds.), *Computational Collective Intelligence. Semantic Web, Social Networks and Multiagent Systems*, Volume 5796, Chapter 65, pp. 743–751. Berlin, Heidelberg: Springer Berlin Heidelberg.

Chuttur, M. (2009). Overview of the technology acceptance model: Origins, developments and future directions. Sprouts: Working Papers on Information Systems 9(37), Indiana University, USA.

Clegg, D. and R. Barker (1994, May). *Case Method, Fast-Track: Fast-Track - A RAD Approach*. Amsterdam, The Netherlands: Addison-Wesley Longman.

Coase, R. (1937). The nature of the firm. *Economica 4*(16), 386–405.

Cockburn, A. (2004, Ostober). *Crystal Clear: A Human-Powered Methodology for Small Teams* (1 ed.). Addison-Wesley Professional.

Cockburn, A. (2005, June). The crystal family of methodologies for software development. http://alistair.cockburn.us/Articles, last accessed 2010/03/19.

Cockburn, A. (2008, June). A user story is the title of one scenario whereas a use case is the contents of multiple scenarios. http://ac.cockburn.us/1610, last accessed 2010/03/17.

Cohen, D., M. Lindvall, and P. Costa (2004). *An Introduction to Agile Methods*, Volume 62 of *Advances in Computers*, pp. 1–66. Elsevier.

Cohn, M. (2004, March). *User Stories Applied: For Agile Software Development*. Addison-Wesley Professional.

Cohn, M. (2005). *Agile Estimating and Planning*. Upper Saddle River, NJ, USA: Prentice Hall PTR.

Cohn, M. (2009). Task boards. http://www.mountaingoatsoftware.com/pages/17-training-for-scrum-task-board-use, last accessed 2010/03/17.

Collins, J., R. Arunachalam, N. Sadeh, J. Ericsson, N. Finne, and S. Janson (2005, November). The supply chain management game for the 2006 trading agent competition. Technical Report CMU-ISRI-05-132, Carnegie Mellon University, Pittsburgh, PA.

Collins, J., W. Ketter, and A. Pakanati (2009, July). An experiment management framework for TAC SCM agent evaluation. In *Workshop: Trading Agent Design and Analysis (TADA) at Twenty-First International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pp. 9–13. AAAI: AAAI Press.

Commission, E. (2009a, October). Investing in the development of low carbon technologies (SET-plan). Commission Documents COM(2009) 519 final, Commission of the European Communities.

Commission, E. (2009b, October). Investing in the development of low carbon technologies (SET-plan): A technology roadmap. Commission Staff Working Document SEC(2009) 1295, Commission of the European Communities.

Conzelmann, G., M. North, G. Boyd, R. R. Cirillo, V. Koritarov, C. M. Macal, P. R. Thimmapuram, and T. D. Veselka (2004). Simulating strategic market behavior using an agent-based modeling approach. In *6th IAEE European Energy Conference on Modeling in Energy Economics and Policy, Zurich*.

Copeland, L. (2001, December). Extreme programming. http://www.computerworld.com/s/article/66192, last retrieved 2010/03/10.

Coplien, J. O. (1994). Borland software craftsmanship: A new look at process, quality and productivity. Technical report, AT&T Bell Laboratories, Orlando, Florida.

Costa, A. C. (2003). Work team trust and effectiveness. *Personnel Review 32*(5), 605 – 622.

Cramton, P. (2003). Electricity market design: The good, the bad, and the ugly. *Hawaii International Conference on System Sciences 2*, 54b+.

Crane, D. and P. McCarthy (2008). *Comet and Reverse Ajax: The Next Generation Ajax 2.0*. Springer.

Czernohous, C. (2005). Simulations for evaluating electronic markets — an agent-based environment. In *SAINT-W '05: Proceedings of the 2005 Symposium on Applications and the Internet Workshops*, Washington, DC, USA, pp. 392–395. IEEE Computer Society.

Davenport, T. H. and L. Prusak (2000). *Working Knowledge* (2 ed.). Harvard Business School Press.

Davis, F., R. Bagozzi, and P. Warshaw (1989). User acceptance of computer technology: a comparison of two theoretical models. *Management science 35*(8), 982–1003.

DeGrace, P. and L. H. Stahl (1990, May). *Wicked Problems, Righteous Solutions: A Catolog of Modern Engineering Paradigms*. Yourdon Press Computing Series. Englewood Cliffs, N.J.: Prentice Hall.

Deindl, M., C. Block, R. Vahidov, and D. Neumann (2008). Load shifting agents for automated demand side management in micro energy grids. In *2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems*.

Dietz, B., K.-H. Ahlert, and C. Block (2010). Driving profile generator. *Social Science Research Network Working Paper Series*.

Dröschel, W. and M. Wiemers (1999). *Das V-Modell 97. Der Standard für die Entwicklung von IT-Systemen mit Anleitung für den Praxiseinsatz*. Munich, Germany: Oldenbourg.

Dröschel, W. and M. Wiemers (2000). *Das V-Modell 97*. Oldenbourg.

Duffy, J. (2006). Agent-based models and human subject experiments. In L. Tesfatsion and K. Judd (Eds.), *Handbook of Computational Economics*, pp. 949–1011. Elsevier.

Dumas, M., B. Benatallah, N. Russell, and M. Spork (2004). A configurable matchmaking framework for electronic marketplaces. *Electronic Commerce Research and Applications 3*(1), 95 – 106.

Dyba, T. and T. Dingsoyr (2008, August). Empirical studies of agile software development: A systematic review. *Information and Software Technology 50*(9-10), 833–859.

Elbow, P. (1998, Jnue). *Writing without Teachers* (2 ed.). Oxford University Press.

Forgy, C. L. (1982). Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence 19*(1), 17–37.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides (1995). *Design Patterns*. Addison-Wesley Professional.

Garicano, L. and S. N. Kaplan (2001). The effects of business-to-business e-commerce on transaction costs. *The Journal of Industrial Economics 49*(4), 463–485.

Gimpel, H. (2006). *Possession, Obsession, and Concession: Preferences and Attachment in Negotiations*. Ph. D. thesis, Universität Karlsruhe (TH).

Gjerstad, S. and J. Dickhaut (2001, April). Price formation in double auctions. In J. Liu and Y. Ye (Eds.), *E-Commerce Agents*, Volume 2033 of *Lecture Notes in Computer Science*, Chapter 7, pp. 106–134. Berlin, Heidelberg: Springer Berlin Heidelberg.

Gladden, G. R. (1982). Stop the life-cycle, i want to get off. *SIGSOFT Softw. Eng. Notes 7*(2), 35–39.

Gloger, B. (2008, April). *Scrum – Produkte zuverlässig und schnell entwickeln* (1 ed.). Munich, Germany: Hanser.

Gloger, B. (2009, July). *Scrum – Produkte zuverlässig und schnell entwickeln* (2 ed.). Munich, Germany: Hanser.

Gode, D. and S. Sunder (1993). Allocative efficiency of markets with zero-intelligence traders: Market as a partial substitute for individual rationality. *Journal of Political Economy 101*(1), 119–137.

Grady, R. B. (1994). Successfully applying software metrics. *Computer 27*(9), 18–25.

Graefe, A. and C. Weinhardt (2008, September). Long-term forecasting with prediction markets a field experiment on applicability and expert confidence. *The Journal of Prediction Markets*, 71–91.

Grenning, J. (2002). Planning poker or how to avoid analysis paralysis while release planning. Technical report, Renaissance Software Consulting.

Hammerstrom, D. J., R. Ambrosio, J. Brous, T. A. Carlon, D. P. Chassin, J. G. DeSteese, R. T. Guttromson, G. R. Horst, O. M. Järvegren, R. Kajfasz, S. Katipamula, L. Kiesling, N. T. Le, P. Michie, T. V. Oliver, R. G. Pratt, S. Thompson, and M. Yao (2007, October). Pacific northwest gridwise testbed demonstration projects: The Olympic Peninsula project. Final report, Pacific Northwest National Laboratory, Richland, Washington 99352.

Hatziargyriou, N., H. Asano, R. Iravani, and C. Marnay (2007, July). Microgrids: An overview of ongoing research, development, and demonstration projects. Berkeley Lab Publications LBNL-62937, Ernest Orlando Lawrence Berkeley National Laboratory.

Hatziargyriou, N. D., A. Dimeas, A. G. Tsikalakis, Pecas, G. Kariniotakis, and J. Oyarzabal (2006, September). Management of microgrids in market environment. *International Journal of Distributed Energy Resources 2*(3), 177–193.

Hendershott, T. J. and R. Riordan (2009, September). Algorithmic trading and information. *Social Science Research Network Working Paper Series*.

Hevner, A., S. March, J. Park, and S. Ram (2004). Design science in information systems research. *Mis Quarterly*, 75–105.

Hirsch, C., L. Hillemacher, C. Block, A. Schuller, and D. Moest (2010). Simulation studies within the meregio field experiment. *it - Information Technology forthcoming*.

Howe, J. (2006). The rise of crowdsourcing. *Wired Magazine 14*(6), 1–4.

Howes, T., G. Good, and M. Smith (1998). *Understanding and deploying LDAP directory services*. Alpel Publishing.

Hrastinski, S., N. Z. Kviselius, H. K. Ozan, and M. Edenius (2010). A review of technologies for open innovation: Characteristics and future trends. *Hawaii International Conference on System Sciences 0*, 1–10.

Hutton, D. W. (1994). *The Change Agents' Handbook: A Survival Guide for Quality Improvement*. ASQ Quality Press.

IEEE (1990). Ieee standard glossary of software engineering terminology. IEEE std 610.12-1990, Institute of Electrical and Electronics Engineers.

Jackson, M. (2000). Mechanism theory. *The Encyclopedia of Life Support Systems*.

Jansen, J., A. van der Welle, and F. Nieuwenhout (2009). The virtual power plant concept from an economic perspective: updated final report. Fenix Techreport Deliverable D3.2.4, Energy Research Centre of the Netherlands (ECN).

Jeffries, R. (2001, August). Essential XP: Card, Conversation, Confirmation. http://xprogramming.com/articles/expcardconversationconfirmation/, last accessed 2010/03/17.

Jennings, N. (2000). On agent-based software engineering. *Artificial Intelligence 117*(2), 277–296.

Jin, C., J. Carbonell, and P. Hayes (2005). Argus: Rete + dbms = efficient persistent profile matching on large-volume data streams. In M.-S. Hacid, Z. W. Ras, and S. Tsumoto (Eds.), *Foundations of Intelligent Systems*, Lecture Notes in Computer Science, pp. 142–152. Springer.

Jonker, C. M. and J. Treur (2001). An agent architecture for multi-attribute negotiation. In *International joint conference on artificial intelligence*, pp. 1195–1201.

Jordan, P. R., B. Cassell, L. F. Callender, and M. P. Wellman (2009). The ad auctions game for the 2009 trading agent competition. Technical report, University of Michigan, Department of Computer Science and Engineering.

Jordan, P. R., C. Kiekintveld, and M. P. Wellman (2007, May). Empirical game-theoretic analysis of the TAC supply chain game. pp. 1188–1195.

Joskow, P. and E. Kahn (2001). A quantitative analysis of pricing behavior in California's wholesale electricity market during summer 2000. NBER Working Paper Series 8157, National Bureau of Economic Research.

Kafura, D. and G. R. Reddy (1987). The use of software complexity metrics in software maintenance. *IEEE Trans. Softw. Eng. 13*(3), 335–343.

Kaldik, M. and M. Eisenblaetter (2010, 03). Internet shopping continues to gain ground. Press Release March 3, 2010, GFK SE, Nuremberg, Germany.

Kambil, A. and E. van Heck (1998). Reengineering the Dutch flower auctions: A framework for analyzing exchange organizations. *Information Systems Research 9*(1), 1–19.

Kambil, A. and E. van Heck (2002). *Making Markets: How Firms Can Design and Profit from Online Auctions and Exchanges.* Harvard Business School Press.

Kan, S. (2002). *Metrics and models in software quality engineering.* Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.

Kano, N., N. Seraku, F. Takahashi, and S. Tsuji (1994, April). Attractive quality and must-be quality. *Journal of the Japanese Society for Quality Control 14*(2), 39–48.

Katok, E. and A. E. Roth (2004). Auctions of homogeneous goods with increasing returns: Experimental comparison of alternative "dutch" auctions. *Management Science 50*(8), 1044–1063.

Kay, J. (2010). *Obliquity.* London, UK: Profile Books Ltd.

Keil, M. and E. Carmel (1995). Customer-developer links in software development. *Commun. ACM 38*(5), 33–44.

Kelly, D. and J. Teevan (2003). Implicit feedback for inferring user preference: a bibliography. *SIGIR Forum 37*(2), 18–28.

Kersten, G. E. and H. Lai (2006). Negotiation support and e-negotiation systems. Internet Research Paper Series INR13/06, InterNeg Research Center, Montreal, Canada.

Ketter, W., J. Collins, M. Gini, A. Gupta, and P. Schrater (2009). Detecting and forecasting economic regimes in multi-agent automated exchanges. *47*(4), 307–318.

Kiekintveld, C., J. Miller, P. R. Jordan, L. F. Callender, and M. P. Wellman (2009). Forecasting market prices in a supply chain game. *Electronic Commerce Research and Applications 8*(2), 63–77. Special Section: Supply Chain Trading Agent Research.

Klemperer, P. (2002a, May). How (not) to run auctions: The european 3g telecom auctions. *European Economic Review 46*(4-5), 829–845.

Klemperer, P. (2002b). What really matters in auction design. *The Journal of Economic Perspectives 16*(1), 169–189.

Kok, J. K., C. J. Warmer, and I. G. Kamphuis (2005). Powermatcher: multiagent control in the electricity infrastructure. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, New York, NY, USA, pp. 75–82. ACM.

Kolitz, K. (2008). *Systemdesign im Market-Engineering - Experimente zu Teilnehmerverhalten und Technologieakzeptanz.* Number 8 in Studies on eOrganisation and Market Engineering. Karlsruhe: Universitätsverlag Karlsruhe.

Kolitz, K., C. Block, and C. Weinhardt (2007). meet2trade: An Electronic Market Platform and Experiment System. In G. E. Kersten, J. Rios, and E. Chen (Eds.), *Group Decision and Negotiation (GDN) 2007*, Montreal, Canada.

Kolitz, K. and D. Neumann (2007). The Impact of Information System Design on eMarketplaces – An Experimental Analysis. In G. E. Kersten, J. Rios, and E. Chen (Eds.), *Group Decision and Negotiation (GDN) 2007*, Montreal, Canada.

Kox, A. (2009, January). Erneuerbare Energien. European Federation of Energy Traders.

Larman, C. and V. R. Basili (2003, June). Iterative and incremental development: a brief history. *Computer 36*(6), 47–56.

Lasseter, R., A. Akhil, C. Marnay, J. Stephens, J. Dagle, R. Guttromson, S. A. Meliopoulous, R. Yinger, and J. Eto (2002, April). Integration of distributed energy resources: The certs microgrid concept. Technical report, Consortium for Electric Reliability Technology Solutions.

Lee, H. G. and T. H. Clark (1996). Impacts of the electronic marketplace on transaction cost and market structure. *Int. J. Electron. Commerce 1*(1), 127–149.

Louis, R. (2006). *Custom Kanban: designing the system to meet the needs of your environment.* Productivity Press.

Luckner, S., F. Kratzer, and C. Weinhardt (2005). STOCCER - A Forecasting Market for the FIFA World Cup 2006. In *Proceedings of the 4th Workshop on e-Business (WEB 2005)*, Las Vegas, USA, pp. 399–405.

MacCormack, A. (2001). Product-development practices that work: How internet companies build software. *MIT Sloan Management Review 40*(2).

MacKie-Mason, J. and M. Wellman (2006). Automated markets and trading agents. In L. Tesfatsion and K. Judd (Eds.), *Handbook of computational economics*, Volume 2, pp. 1381–1431. Elsevier.

Mäkiö, J. and I. Weber (2004). Component-based Specification and Composition of Market Structures. In M. Bichler and C. Holtmann (Eds.), *Coordination and Agent Technology in Value Networks*, pp. 127 – 137. Berlin, Germany: Gito.

Mäkiö, J. and I. Weber (2005). Modeling approach for auction based markets. *Applications and the Internet Workshops, IEEE/IPSJ International Symposium on 0*, 400–403.

Malone, T. W., J. Yates, and R. I. Benjamin (1987, June). Electronic markets and electronic hierarchies. *Commun. ACM 30*(6), 484–497.

Marks, R. (2006). Market design using agent-based models. In L. Tesfatsion and K. Judd (Eds.), *Handbook of Computational Economics*, pp. 1339–1380. Elsevier.

McAfee, R. et al. (1998). Four issues in auctions and market design. *Revista de Análisis Económico 13*(1), 7–24.

McBreen, P. (2002, July). *Questioning Extreme Programming.* Addison Wesley.

McCabe, T. (1976). A complexity measure. *IEEE Transactions on Software Engineering 2*(4), 308–320.

McConnell, S. (1996, July). *Rapid Development: Taming Wild Software Schedules.* Microsoft Press.

McCracken, D. D. and M. A. Jackson (1982). Life cycle concept considered harmful. *SIGSOFT Softw. Eng. Notes 7*(2), 29–32.

McMillan, J. (2003). *Reinventing the bazaar: A natural history of markets.* WW Norton & Company.

Meeus, L. and R. Belmans (2007). Is the prevailing wholesale market design in europe and north america comparable? In *Power Engineering Society General Meeting, 2007. IEEE*, pp. 1–5.

Meier, U. (2004, July). Das schlafverhalten der deutschen bevölkerung – eine repräsentative studie. *Somnologie 8*(3), 87–94.

Miller, J., S. Page, and B. LeBaron (2007). *Complex adaptive systems: An introduction to computational models of social life.* Princeton University Press Princeton and Oxford.

Miranker, D. P. (1987). Treat: A better match algorithm for ai production system matching. In *AAAI*, pp. 42–47.

Monson-Haefel, R. and A. Weissinger (2003). *Enterprise JavaBeans.* O'Reilly & Associates, Inc. Sebastopol, CA, USA.

Myerson, R. B. and M. A. Satterthwaite (1983, April). Efficient mechanisms for bilateral trading. *Journal of Economic Theory 29*(2), 265–281.

Neumann, D. (2007). *Market Engineering - A Structured Design Process for Electronic Markets.* Universitätsverlag Karlsruhe.

Neumann, D., C. Block, C. Weinhardt, and Y. Karabulut (2007, June). Knowledge-Driven Selection of Market Mechanisms in E-Procurement. In *Proceedings of the 15th European Conference on Information Systems (ECIS' 07)*, St. Gallen, Switzerland, pp. 143–154.

Nicolaisen, J., V. Petrov, and L. Tesfatsion (2001). Market power and efficiency in a computational electricity market with discriminatory double-auction pricing. *Evolutionary Computation, IEEE Transactions on 5*(5), 504–523.

Nisan, N. and A. Ronen (2001, April). Algorithmic mechanism design,. *Games and Economic Behavior 35*(1-2), 166–196.

Niu, J., K. Cai, S. Parsons, P. McBurney, and E. Gerding (2010). What the 2007 TAC Market Design Game tells us about effective auction mechanisms. *Autonomous Agents and Multi-Agent Systems. Special Issue on Market-Based Control of Complex Computational Systems*.

North, M., G. Conzelmann, V. Koritarov, C. Macal, P. Thimmapuram, and T. Veselka (2002). E-laboratories: agent-based modeling of electricity markets. In *2002 American Power Conference*, pp. 1–19.

Orey, S. and W. Pruitt (1973). Sample functions of the n-parameter wiener process. *The Annals of Probability 1*(1), 138–163.

Osborne, L., J. Brummond, R. Hart, M. Zarean, and S. Conger (2005, October). Clarus: Concept of operations. Technical Report FHWA-JPO-05-072, U.S. Department of Transportation.

Pahl, G. and W. Beitz (1984). *Engineering Design.* Bath, UK: The Pitman Press.

Phelps, S. G. (2007, July). *Evolutionary Mechanism Design.* Ph. D. thesis, University of Liverpool.

Poppendieck, M. and T. Poppendieck (2006, September). *Implementing Lean Software Development: From Concept to Cash* (1 ed.). Addison-Wesley Professional.

Porter, B. W., R. Bareiss, and R. C. Holte (1993). *Readings in Knowledge Acquisition and Learning: Automating the Construction and Improvement of Expert Systems.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. 1-55860-163-5.

Prechelt, L., B. Unger, and W. Tichy (1999). A controlled experiment on inheritance depth as a cost factor for maintenance. *IEEE Trans. on Software Engineering 65*, 115–126.

Program, U. C. C. T. (2006, September). Technology options for the near and long term. Technical report, U.S. Department of Energy.

Rachlevsky-Reich, B., I. Ben-Shaul, N. Chan, A. Lo, and T. Poggio (1999). GEM: a global electronic market system. *Information Systems 24*(6), 495–518.

Recordon, D. and D. Reed (2006). Openid 2.0: a platform for user-centric identity management. In *DIM '06: Proceedings of the second ACM workshop on Digital identity management*, New York, NY, USA, pp. 11–16. ACM.

Rittel, H. W. J. and M. M. Webber (1973, June). Dilemmas in a general theory of planning. *Policy Sciences 4*(2), 155–169.

Royce, W. (1970). Managing the development of large software systems. In *Proceedings of IEEE Wescon*, Volume 26, pp. 1–9.

Sanchez, I. (2006, January). Short-term prediction of wind energy production. *International Journal of Forecasting 22*(1), 43–56.

Schaeffer, G. J. and H. Akkermans (2006, July). Distributed intelligence in critical infrastructures for sustainable power: Final summary report. Project Deliverable ECN FSP WP53 002, Energy Research Centre of the Netherlands (ECN).

Schmid, B. (1997). Requirements for electronic markets architecture. *Electronic Markets 7*(1), 3–6.

Schönfeld, A. and C. A. Block (2010, May). A meta-framework for agile development of soa market platforms. *Social Science Research Network Working Paper Series*.

Schüller, C. and W. Wörndl (2008, September). Automated user feedback generation in the software development of mobile applications. In J. Roth (Ed.), *5. GI/ITG KuVS Fachgespräch Ortsbezogene Anwendungen und Dienste*, Volume 42 of *Schriftenreihe der Georg-Simon-Ohm-Hochschule Nürnberg*.

Schwaber, K. (2009, January). What is Scrum? http://www.scrumalliance.org/resources/227, retrieved 2010/03/10.

Schwaber, K. and M. Beedle (2001, October). *Agile Software Development with Scrum*. Upper Saddle River: Prentice Hall.

ScrumAlliance (2010). What Is Scrum? http://www.scrumalliance.org/learn_about_scrum.

Segev, A., J. Gebauer, and F. Färber (1999). Internet-based electronic markets. *Electronic Markets 9*(3), 138–146.

Seifert, S. and K.-M. Ehrhart (2005). Design of the 3G Spectrum Auctions in the UK and in Germany: An Experimental Analysis. *German Economic Review 6*(2), 229 – 248.

Selby, R. (2007). *Software engineering: Barry W. Boehm's lifetime contributions to software development, management, and research.* Wiley-IEEE Computer Society Pr.

Selhorst, S. (2006). Prioritizing software requirements with kano analysis. *The Pragmatic Marketer Magazine 4*(3), 24 – 29.

Shepperd, M. (1988). A critique of cyclomatic complexity as a software metric. *Softw. Eng. J. 3*(2), 30–36.

Shingo, S. (1989, July). *A Study of the Toyota Production System* (Rev Sub ed.). Productivity Press.

Siddiqui, A., E. Bartholomew, and C. Marnay (2004, July). Empirical analysis of the spot market implications of price-elastic demand. Berkeley Lab Publications LBNL-56141.

Smith, V. (1982). Microeconomic systems as an experimental science. *The American Economic Review 72*(5), 923–955.

Sodomka, E., J. Collins, and M. Gini (2007). Efficient statistical methods for evaluating trading agent performance. pp. 770–775.

Spees, K. and L. Lave (2008). Impacts of responsive load in PJM: Load shifting and real time pricing. *The Energy Journal 29*(2), 101–122.

Spolsky, J. (2005, July). Hitting the high notes. http://www.joelonsoftware.com/articles/HighNotes.html, last accessed 2010/03/17.

Standish Group (1995). Chaos. Technical report, The Standish Group.

StatBA (2008). Monatsbericht über die Elektrizitätsversorgung. Technical report, Statistisches Bundesamt, Wiesbaden, Germany.

StatBA (2009, November, 26). Umsatzvolumen über elektronischen handel steigt. Pressemitteilung 453, Statistisches Bundesamt, Bonn.

Stathel, S., S. Luckner, F. Teschner, C. Weinhardt, A. Reeson, and S. Whitten (2009). Akx – an exchange for predicting water dam levels in australia. In *Information Technologies in Environmental Engineering*, Chapter 6, pp. 78–90. Berlin, Heidelberg: Springer Berlin Heidelberg.

Stathel, S., S. Luckner, and C. van Dinther (2008). Information Efficiency and Liquidity in Information Markets - A market maker based approach, Vortrag. In *Third Workshop on Prediction Markets, ACM Conference on Electronic Commerce 2008*, Chicago, USA.

Stephens, M. and D. Rosenberg (2003, August). *Extreme Programming Refactored: The Case Against XP* (1 ed.). Apress.

Stone, P. (2003). Multiagent competitions and research: Lessons from RoboCup and TAC. In G. A. Kaminka, P. U. Lima, and R. Rojas (Eds.), *RoboCup-2002: Robot Soccer World Cup VI*, pp. 224–237. Berlin: Springer Verlag.

Strecker, S. and S. Seifert (2003). Preference revelation in multi-attribute bidding procedures: an experimental analysis. In *14th International Workshop on Database and Expert Systems Applications, 2003. Proceedings.*, pp. 850–854. IEEE Comput. Soc.

Sueyoshi, T. and G. Tadiparthi (2008). An agent-based decision support system for wholesale electricity market. *Decision Support Systems 44*(2), 425–446.

Sun, J. and L. Tesfatsion (2007). Dynamic testing of wholesale power market designs: An open-source agent-based framework. *Computational Economics 30*(3), 291–327.

SWEBOK (2004). *Guide to the Software Engineering Body of Knowledge (SWEBOK).* IEEE Computer Society.

Takeuchi, H. and I. Nonaka (1986). The new new product development game. *Harvard Business Review 64*(1), 137–146.

Tamma, V., S. Phelps, I. Dickinson, and M. Wooldridge (2005, March). Ontologies for supporting negotiation in e-commerce. *Engineering Applications of Artificial Intelligence 18*(2), 223–236.

Taylor, D. and P. Wurman (2000). An xml schema for the parameterization of auctions. Technical report, Intelligent Commerce Research Group, North Carolina State University.

Teschner, F. and A. Storkenmaier (2010). Short Selling in Prediction Markets. Proceedings of the 11th Group Decision and Negotiation Conference (GDN), Extended Abstract.

Tesfatsion, L. (2002). Agent-based computational economics: Growing economies from the bottom up. *8*(1), 55–82.

Tsvetovatyy, M., M. Gini, B. Mobasher, and Z. W. Ski (1997). Magma an agent based virtual market for electronic commerce. *Applied Artificial Intelligence: An International Journal 11*(6), 501–523.

van Dinther, C. (2006, August). *Adaptive Bidding in Single Sided Auctions under Uncertainty – An Agent-based Approach in Market Engineering.* Ph. D. thesis, Universität Karlsruhe (TH), Karlsruhe, Germany.

van Dinther, C., A. Weidlich, and C. Block (2006). Energiemaerkte der zukunft. White paper, Universitaet Karlsruhe (TH).

Vandenpoel, D. and W. Buckinx (2005, October). Predicting online-purchasing behaviour. *European Journal of Operational Research 166*(2), 557–575.

Varga, L. Z., N. R. Jennings, and D. Cockburn (1994). Integrating intelligent systems into a cooperating community for electricity distribution management. *Int. Journal of Expert Systems with Applications 7*(4), 563–579.

Veit, D., A. Weidlich, and J. A. Krafft (2009). An agent-based analysis of the german electricity market with transmission capacity constraints. *Energy Policy 37*(10), 4132–4144.

Venkatesh, V., M. Morris, G. Davis, and F. Davis (2003). User Acceptance of Information Technology: Toward a Unified View. *MIS Quarterly 27*(3), 425–478.

Verrecchia, R. (1979). On the theory of market information efficiency. *Journal of Accounting and Economics 1*(1), 77–90.

von Dollen, D. (2009, June). Report to NIST on the smart grid interoperability standards roadmap. Technical Report SB1341-09-CN-0031, Electric Power Research Institute (EPRI).

Vytelingum, P., T. D. Voice, S. D. Ramchurn, A. Rogers, and N. R. Jennings (2010). Agent-based micro-storage management for the smart grid.

Wang, S., S. Zheng, L. Xu, D. Li, and H. Meng (2008, November). A literature review of electronic marketplace research: Themes, theories and an integrative framework. *Information Systems Frontiers 10*(5), 555–571.

Weidlich, A. (2008). *Engineering Interrelated Electricity Markets – An Agent-Based Computational Approach.* Ph. D. thesis, Universität Karlsruhe (TH).

Weidlich, A. and D. Veit (2008). A critical survey of agent-based wholesale electricity market models. *Energy Economics 30*(4), 1728–1759.

Weinhardt, C., C. v. Dinther, M. Grunenberg, K. Kolitz, M. Kunzelmann, J. Mäkiö, I. Weber, and H. Weltzien (2006). *CAME-Toolsuite meet2trade - auf dem Weg zum Computer Aided Market Engineering. Abschlussbericht des Projekts Electronic Financial Brokerage als wissensintensive Dienstleistung - ein generischer Ansatz (EFB).* Number 3 in Studies on eOrganisation and Market Engineering. Karlsruhe: Universitätsverlag Karlsruhe.

Weinhardt, C., D. Neumann, and C. Holtmann (2003). Market engineering. *Wirtschaftsinformatik 45*(6), 635–640.

Weinhardt, C., D. Neumann, and C. Holtmann (2006). Computer-aided market engineering. *Communications of the ACM 49*(7), 79.

Wellman, M. P., A. Greenwald, and P. Stone (2007). *Autonomous Bidding Agents.* MIT Press.

West, D. and T. Grant (2010, January). Agile development: Mainstream adoption has changed agility – trends in real-world adoption of agile methods. Technical report, Forrester Research.

Wiegers, K. (2000). Karl Wiegers describes 10 requirements traps to avoid. *Software Testing & Quality Engineering 2*(1).

Wirdemann, R. (2009). *Scrum mit User Stories*. Munich, Germany: Hanser.

Womack, J. P., D. T. Jones, and D. Roos (1991, November). *The Machine That Changed the World: The Story of Lean Production* (Reprint ed.). HarperPaperbacks.

Wooldridge, M. and N. Jennings (1995). Intelligent agents: Theory and practice. *Knowledge engineering review 10*(2), 115–152.

Wurman, P., M. Wellman, W. Walsh, and K. O'Malley (1999). Control architecture for a flexible Internet auction server. In *Proceedings of the First IAC Workshop on Internet Based Negotiation Technologies*, Yorktown Heights, New York.

Wurman, P. R. (1999). *Market structure and multidimensional auction design for computational economies*. Ph. D. thesis, University of Michigan.

Wurman, P. R., W. E. Walsh, and M. P. Wellman (1998, November). Flexible double auctions for electionic commerce: theory and implementation. *Decis. Support Syst. 24*(1), 17–27.

Zhou, Z., W. Chan, and J. Chow (2007). Agent-based simulation of electricity markets: a survey of tools. *Artificial Intelligence Review 28*(4), 305–342.