Conceptualization and Realization of Agile Building Blocks for a User-Configurable Trading Platform

Moritz C. Weber
E-Finance Lab at the Goethe University Frankfurt
moweber@wiwi.uni-frankfurt.de

Abstract
Market models are the blue prints of electronic trading platforms. Having various specifications and trading styles, traders on today’s market places are challenged to establish to find markets with models that fit their trading habits. Established market structures will need to become more flexible in order to address the needs of traders. This paper analyses efficient building blocks for such a user-configurable trading platform based on the design science research methodology. After investigating previous academic work on trading platforms, an agile trading platform is designed to be flexible, fast and to have low entrance barriers. We present a prototype as a proof-of-concept and subsequently evaluate and discuss it with given kernel theories.

1. Introduction

Markets are meant to identify efficient allocations of goods. Therefore market participants have to estimate their individual utility of these goods and agree on an allocation mechanism that fits their needs. As in real-world settings there are typically many alternative allocations, determination of efficient ones can become a complex problem. To foster efficient allocation diverse organizational setups or coordination mechanisms have evolved, ranging from pure hierarchical approaches, where a central authority affects resource reallocation via defined processes, to pure market-driven organizations, where market participants use a decentralized price system to signal their individual preferences and opportunities [1].

Although a market’s price system provides an efficient incentive system, effective communication and high flexibility, there are situations where participants prefer hierarchical organization rather than a market driven organization. This is typically the case when there is a high asset specificity and a low transaction frequency [2], e.g. as it is a unique transaction of a non-standardized good. Due to the specificity, a hierarchical coordination is more effective, as individual knowledge and preferences can be expressed more easily. However, as it is associated with higher search and negotiation costs, it may be suboptimal and lead to bilateral negotiation and agreements. Market coordination on the other hand offers its greatest advantages with respect to standardized goods that are traded more frequently. Common to these approaches is that market participants reveal their transaction intentions by specifying their reservation price and (if applicable) the amount of goods that shall be transacted (quantity). While this two-dimensional revelation of preferences is easy to understand it also limits the expressivity of complex user preferences. To cater for more complex needs of users, market organizers introduced numerous advanced order types e.g. orders that only become active once a certain price level has been reached (stop orders) or orders that are automatically deleted once another order has been executed. However, such order types are only another restrained vehicle to express user preferences. Regardless of how many order types are offered, there will always be user preferences that cannot be completely reflected by a standardized order. This few-size-fits-all approaches with respect to expressivity of preferences by two-dimensional orders may not cater all heterogeneous users’ needs sufficiently, but will leave many users dissatisfied [23]. Hence, new and innovative concepts and platforms are required that provide flexibility in how individual preferences are expressed and eventually matched to each other. The expression of preferences does not have to be constrained to reservation price and quantity. A market participant may also specify further preferences e.g. with respect to its counterparty or the ranking of potential allocations. With this individuality new markets are established, as a market is defined by a group of participants that accepts the same rules. This user-driven approach towards collaborative innovations requires a platform that is customizable to the heterogeneous context of the users. Hence, the research at hand aims at investigating efficient structures for such a mass-customizable market...
platform. Therefore within the paper a platform concept is designed and evaluated according to research and best practice principles that constitutes an agile market model (AMM).

The remainder is structured as follows: Section 2 provides an overview of the state of research and introduces design science research as our methodology. Subsequently, section 3 derives the requirements for an AMM from the data, functional and architectural perspective. Section 4 highlights the specific AMM architecture and implementation, which is subsequently evaluated and discussed in section 5, before the paper concludes with an outlook.

2. State of research

In the following subsection related research is provided on which this paper is based or relates to. The subsection 2.1 sets the abstract domain context for our research before 2.2 highlights related work in this area. The subsection 2.3 details the theoretical foundation of design science and introduces the related kernel theories used for latter evaluation.

2.1 Flexible and dynamic market models

Typically, (electronic) negotiations are based on explicit rules that govern the available ways of communication and determine whether and under which conditions a transaction is conducted. In real-world settings different sets of rules have established, each addressing different needs of market participants, market organizers as well as requirements set by the specific type(s) of goods that are at the heart of the potential transactions. Key characteristics of the potential organization forms of electronic negotiations and systems are highlighted by the Montreal Taxonomy [4].

Auction theory distinguishes between one-sided auctions and double auctions [35]. While in the former there is a 1:n relationship between buyers and sellers (or vice versa), the double auction is characterized by a n:m relationship between buyers and sellers. For one-sided auctions some standard auction mechanisms have evolved. E.g. the English Auction as used in art auctions, where the highest bid wins, or the Dutch Auction, as used for selling flowers, where the first bid wins that accepts the price from a descending tickers. In double-sided auctions, market participants have to specify the trading interests in the form of buy and sell orders. These are submitted to a so-called central limit order book (CLOB) [46].

The further handling of orders depends on the rules set out by a market, which can be categorized along the following structural dimensions [7]:

- **Transparency** determines whether all, none or only the best buy and sell orders are displayed to market participants.
- **Price determination** defines which rules are followed to determine transaction prices and which orders are to be executed first (order matching). This includes quote-driven approaches, where orders are only executable versus firm buy and sell prices that are simultaneously posted by a market maker. In contrast to this is the order-driven approach, where orders can interact directly with each other according to the rules and order priorities of a specific trading mechanism.
- **Trading frequency** defines whether order execution is triggered continuously (continuous double auction) or following a pre-defined schedule (call auction). In the former case an order is executed once it becomes executable versus another order according to the rules of price determination. This yields discriminatory pricing. In the latter case orders are collected for some time (call phase) without execution. At scheduled points in time all executable orders are executed in a batch at a uniform price.
- **Firmness** defines whether posted limits are binding or are only indication of interest. In the latter case this would open up potential for further (bilateral) negotiations. Moreover, this would harm applicable price determination rules.

According to [57], a **static market model** is defined by a particular specification of the structural dimensions, i.e. exactly one parameter value. Flexible market models are characterized by a market structure where the parameter values can change over time, following pre-defined rules. E.g. a continuous double auction, that is surrounded by an opening and closing call auction that are held at fixed times. In contrast to these setups, [57] define dynamic market models as markets where market participants can define their desired parameter values along the structural dimensions for particular transactions.

A limit order book is a central component of a static market model [6]. To foster a certain kind of fairness in a market, specific rules, like non-discretionary order handling, are enforced to guarantee a fair execution of orders from different market participants. Therefore priorities can be enforced to sort orders in a defined sequence. A “Limit-time-priority” (LTP) rule is an example for such a priority. It enforces that incoming orders in a continuous double auction are immediately checked for executability against orders residing in the order book. These residing orders are sorted by their given limit and for orders with an identical limit by their arrival time stamp. Thereby, best limit and longest waiting orders are checked for executability against incoming orders. If no execution is possible, incoming orders are
integrated into the order book according to the mentioned rule. Despite its fairness, the LTP limit order book has static order properties and fixed priority and preference rules. That makes it inflexible towards changing requirements of the participants. Even when the static continuous double auction is combined e.g. with call auctions to open and close the market, in such a “flexible” setup it still can be difficult for participants to sketch the specific properties of their goods, their limiting trading rules and their priorities and preferences.

A good overview of possible requirements and criteria for market models is given by the Montreal Taxonomy [64]. It is a collection of exogenous and endogenous criteria for an electronic negotiation process. Exogenous criteria are rules and knowledge from the specific business context or domain-independent criteria like ethical standards. Endogenous criteria are describing the criteria for the process itself. Implicit endogenous criteria can specify soft criteria like fairness or Pareto efficiency. Explicit criteria describe characteristics of the negotiation activities. The taxonomy splits the process into an intention phase (Specification, Submission and Analysis), which describes mostly the creation, modification and cancelation of an order, and an agreement phase that distinguishes activities of matching, allocation and acceptance. Due to the page limitation we highlight the criteria of matching and allocation. Matching has the goal to find pairs of offers that match each other considering scored offers ranked by the preferences of the participants. This can include criteria for “scheduling”, whether the matching is clocked, triggered or continuous, and “evaluation”, whether offers are ranked or just listed. In addition criteria for sorting and resolution can define whether and how trading partners are prioritized and offer conflicts are solved. The allocation criterion defines whether the allocation distribution is (non-)discriminatory, provisions are offer dependent or the allocation configuration is mediated by the system (or open to the participant). In general the Montreal Taxonomy gives a sufficient theoretical overview for pre-definable characteristics in electronic negotiations, but is not intended to be exhaustive.

2.2 Related work

Non-standardized assets, illiquid stocks or block trading are typical scenarios where static and flexible market models are limited [5]. To address these shortcomings of static and also flexible market models, [5] suggest the AMTRAS system based on a dynamic market model. AMTRAS stands for “Agent Mediated Trading System” and its highlight is that the static order book structures are transferred into negotiation rules of trading agents that mediate the trading process on behalf of the market participant. It supports the search process for matching counterparts and the price discovery process. Participants set up a dedicated agent for each order that shall be processed by the system. Agents can be defined with fuzzy order specifications, the search rooms for subgroups of participants. Furthermore it can be parameterized whether the agent shall negotiate with (non-) mandatory orders or a single auction. With the ability to give the participant more options to parameterize how to trade a specific order by a market model that is more dynamic than the discussed static and flexible versions, the system does not allow to define completely new structures, rules and negotiation styles. AMTRAS is fixed in its predefined boundaries, but can be customized flexibly within those by a set of parameters.

MetaMarkets [8] adopts and extends the basic concept of a dynamic market model. Based on the notion that the dynamic market model is restricted to a set of parameters for a given market, MetaMarkets’ [8] goal is a cascading Market Model that allows the configuration and combination of multiple market models. Thereby, two perspectives can be covered: On the one hand a market designer wants to be able to structure and restructure market models due to changing regulatory, competitive or technical requirements. On the other hand investors and their orders shall have more degrees of freedom in how to trade their orders. In consequence MetaMarkets’ cascading market model allows the specification of multiple market models that can be combined in any needed sequential and parallel order, so that market designers and investors can decide how an order shall be passed through multiple markets. From a technical point of view a MetaMarket combines a set of rules and multiple market models into a single entity. The specification of a MetaMarket is set up in an XML configuration file that allows the definition of the market set, information on the sequence of orders and defined relations between the defined market structures. Even if the structure of MetaMarket is more adjustable than the structure of the AMTRAS system, the XML grammar forces users to use predefined parameterizable XML structures. Definitions beyond the given XML tags and their possibilities to parameterize XML attributes make it difficult to define new or more fine-grained market structures.

Based on MetaMarket’s XML configuration, [89] develop a trading system called meet2trade that supports design, configuration, trading and test of/on market models. Meet2trade is designed to be flexible for various business domains and order structures. Predefined market model blocks combined with
predefined trading concepts shall foster fast development and evaluation. The four major components of the software suite are arranged in a three-layer architecture: an adaptive client, the meet2trade Experimental System (MES), the Simulation Environment AMASE and the Market Modeling Language (MML). MML is based on the MetaMarkets XML grammar and results in coarse, predefined market blocks as single/double sided auctions with continuous and call market trading that can be combined sequentially and parallel. Furthermore, bundle and multi-attribute auctions are provided. Meet2trade is basically a very powerful and adjustable auction configuration platform but is limited to its predefined core elements adopted from MetaMarkets. Flexibility and dynamics are restricted to the composition of existing components, which are not agile, changeable or extendable to new innovative requirements of market designers and investors. Thus, we require an agile market model that enables the instantiation of existing but also new and innovative features and structures of market models.

2.3 Design science research and kernel theories

Design is defined as “the use of science principles, technical information and imagination in the definition of a structure, machine or system to perform pre-specified functions with the maximum economy and efficiency” [10]. Design science research (DSR) describes how the artificial can be designed [11] or how IT or IS artifacts can be developed [12]. These artifacts are constructs, models, methods or instantiations that are built, evaluated, theorized and justified [13, 14]. The process of designing an IT/IS artifact and generalized knowledge [15] is described by Information System design theories [12] that define how people and organizational requirements for technology are combined by theoretical guidelines and methodological processes [14, 15] to solve a certain problem. They are driven by theory building [13] and practical requirements [16] like competitive advantages, e.g. in electronic commerce [17].

To build a theoretical foundation, so-called kernel theories are applied from natural and social science [15, 18]. These theories support the deduction of evaluable hypotheses from meta-requirements and a meta-design [12]. Therefore a new IT / IS artifact is designed that allows, based on established theories, to derive a new theory on a given class of practitioner problems. This supports theory building for design and action [19] by solving a problem with guidelines of DSR like [14] or [20]. The design process has two steps. After the design of the artifact it has to be evaluated against predefined meta-requirements and meta-designs [16, 21]. The evaluation reinforces the optimization of the IT artifact and thus the process of designing itself [14, 18].

For our problem at hand, to design and evaluate efficient structures for mass-customizable market platforms we establish a rigorous theoretical setup from DSR guidelines [14, 19, 20]. We begin the design of our IT / IS artifact [12] by giving an overview of well-established kernel theories from Computer Science and Learning Theory to build technically efficient and still human manageable building block structures for a market platform.

Computability theory: In Computer Science the computability theory terms the attribute of universal computability in abstract systems and programming languages Turing-complete [22]. Next to the computability of problems complex systems can be described as higher-order logic. To describe how functions can abstract and apply higher-level logic the so-called Lambda calculus is applied [23]. To keep even complex systems simple, the concept of orthogonality is introduced. It enforces that operations are side-effect free and that each operation does not change more than one thing [24]. This is strongly connected to the UNIX principle which claims that tools should fit to one specific task and that they should interact via text streams as universal interface [25].

Complexity theory: Given a computable problem complexity theory describes the amount of resources that an algorithm consumes. These resources are typical calculated as need of time or memory resources. To ease and abstract cost function and measure the so-called O-/Landau notation [26] is used. It gives measurable orders of magnitudes and neglects linear or constant factors. The O-/Landau notation is used to define certain boundaries of so-called complexity classes [27] as constant (O(1)), logarithmic (O(log n)), squared (O(n^2)) runtime or memory consumption. It can be used to prove how good an algorithm performs in the best case, the worst case and on average.

Learning theory and Constructivism: Design science itself, but also mass-customizable systems for construction, fall in the learning paradigm of Constructivism. Constructivism assumes that the learner constructs a model of his learning problem by a social-interactive learning process. This is strongly connected with cognitive psychology that is often used in computer-based learning systems [29]. The human cognition – especially the short term memory – is limited to a set number of chunks [30]. If this limit is exceeded users get confused or forget about details. To understand and learn even complex systems the learner should not be overwhelmed by the learning subject.
This problem can be overcome by limiting the number of learning chunks per learning or work unit. This is called a gentle slope of complexity [31].

3. Requirements for an agile market model

The agile market model shall address current weaknesses of static, flexible and dynamic market models and frameworks in a holistic and complete manner. Therefore the AMM has to affect three independent but connected aspects: data, functions and architecture. Data represent static aspects of the system like its current state and its changes in a persistent way. Functions describe how the system can change the state of data and a set of functions can build complex activities and processes. Keeping the underlying data structures and functions simple, modular and generic, the architecture has to provide the ability to combine data and functions in a use-case and domain-specific manner.

Taking the project iron triangle [32] as a reference, it is shown that models and systems cannot be perfectly optimized to the concepts of quality, speed and low costs at the same time. Nevertheless, the AMM shall challenge current designs and implementations presented in the previous section. It shall push their boundaries further toward quality, speed and low cost by using innovative concepts. In our case, quality shall be reflected by agility, which is defined as “the ability to both create and respond to change in order to profit in a turbulent business environment” [33]. The concept of speed is covered in our case by the efficiency of the system to process orders of magnitude. Having a high agile and efficient solution we assume that cost of staff and the time of users are the most important factor. Having a high agile and efficient solution we assume that cost of staff and the time of users are the most expensive resource that has to be supported and managed.

As last concept it is required that systems are not only technical agile and efficient but must be understandable and learnable for their users. With easiness to understand, to learn and to use we cover the concept of low cost. That seems to neglect cost of implementing and operation of the AMM, but it is assumable that once the AMM is finally implemented the ability to use it efficiently is the most important factor. Having a high agile and efficient solution we assume that cost of staff and the time of users are the most expensive resource that has to be supported and managed.

Given the previous aspects and concepts, we shed also more light on the weaknesses of prior work by highlighting design research gaps that shall be addressed by the requirements of the AMM.

Data Agility shall be given by not enforcing the use of predefined properties/attributes of goods and bundle structures. AMM shall also not enforce certain fixed data structures like a limit order book. Goods’ data, trading rules, priorities and preferences shall neither be fixed for users nor for user groups or the complete trading system. The users decide about their requirements to the system. The system supports the merge of compatible goods and requirements of users.

Data Efficiency is given by just specifying attributes and properties for data, rules, priorities and preferences that a user, a group or its systems needs. The user, not the system, decides how lightweight his/her data structure shall be. Data structures are not fixed by a schema, but by a meta schema from the AMM.

Data Understandability is provided to the user by the possibility to define own domain-specific structures of goods, rules, priorities and preferences for itself, its groups or its required system.

Functional Agility shall be achieved by a universal and complete set of atomic functions that are general for trade processing. Predefined trading activities and processes like in the Montreal Taxonomy [4, 9] shall not be enforced but possible to be defined. The set of functions shall be extendable by the user on client and server-side depended to his permissions.

Functional Efficiency shall be given by atomic operations that are designed as best of breed solution with a lightweight design and thin interfaces.

Functional Understandability shall be ensured by the UNIX philosophy [25] that each function does one thing and does it well. The number of basic functions should be limited (depending on user) to a reasonable number of functions that are capable for the human cognition [30]. Complexity of the system shall come from the combination of simple functions, not from the design of the system itself.

Architectural Agility shall guarantee that users actually just use those layers and parts of the AMM that they actually need. Not the design of the system enforces which parts and layers are or can be used, but the decision of the user and its permissions.

Architectural Efficiency comes from a combination of the previously required data structures, functions and the Architectural Agility.

Architectural Understandability shall be guaranteed by providing the user a system that fits his needs with a number of used parts and/or layers that are capable for the human cognition in most cases [30]. Accordingly, the extent of documentation required to understand the AMM should be limited.

4. AMM System

In the following the design and implementation of the system is described from the data, the functional and the architectural perspective. To guarantee a universal data object management life cycle, the AMM
Domain objects have a distinction to raw descriptive “data”, limiting “rules” and trading “priorities” and “preferences”. These can be set as key/value-combinations to 5 scopes (good, session, user, group, config). The most basic one is the “good” scope. It represents a good and shall highlight that its asset-specific properties are optional (like price/limit/value and quantity). By defining its descriptive data, limiting rules, good-specific priorities and preference it is set up how a good can be traded with any other offered good. All data is optional and user-defined, except for reserved keywords that are used to configure defined system behaviors. Without any definition the system just offers the ability to swap “things” randomly. If rules are defined then config definitions dominate (overwrite) group definitions. These dominate user definitions which in turn dominate goods definitions (s. Figure 1). Config and group scope aim to cover static and flexible market models. User, session and good scope cover dynamic market models. The AMM does not limit the user to one of these paradigms but supports the full range of paradigms by design. Goods are stored and each value of a property is indexed in a property specific index. This is needed to support dynamic ranking and matching depending on all defined scopes and the related data, rules, priorities and preferences. Thereby we achieve “multi-attributive, user-specified order book views”. Permissions for object definitions depend on POSIX file access permissions [35] that define rights for reading, writing and executing objects for its owner, its associated group and “others”.

Each CRUD operation has one (last) optional parameter that is called “callback”. Callback functions are used for asynchronous request and provide a function that is called when the response of the request is received. An asynchronous function returns immediately and the client never blocks. The callback has a success and an error object as parameter, but just one of them is defined in any given case. The CRUD operations are supplemented with trading specific callback functions and build thereby dynamic and non-blocking “callback processes chains” (s. Figure 2). Trading is always done by the “swap” operation shifting goods conforming to a given allocation, if all participants permit the transaction. The allocation or just the trade data can be persisted optionally by adding a “log” callback operation. To get a tradable allocation the “match” operation can be used that implements a parameterizable universal VCG-mechanism from auction theory [36, 37].

Rankings can be calculated on server-side by a preference-weighted union of all property-value indices for a property of a swap. This union of weighted property sets is intersected with all properties that have defined priorities. So a linear-weighted ranking is generated for all matching swaps corresponding attributes. Thereby we generalized the static CLOB to any definable combination of limiting rules, personal priorities and linear defined preferences.
Sockets [41] for bi-directional real-time communication and HTTP by the “session” API. Users can use and combine APIs at all levels as they need them from a front-end point of view (s. Figure 3). So we call this concept “stackable APIs” instead of multi-layer architecture. Any missing operation or API can be deployed on client or server-side by plugging in to the given APIs and having the needed permissions on needed scopes and data of the system.

AMM is implemented as a running proof-of-concept prototype that aims to shift boundaries of the previous discussed implementations in the dimensions of agility, efficiency, and understandability. Therefore we use an in-memory data structure server called Redis [42]. That allows accessing schemaless key-value-objects, sets, sorted sets as well as unions and intersections of those. Redis is designed and used for high-scalable, high-available and time-critical infrastructures like Yahoo, Twitter, Flicker, Instagram, The Guardian and StackOverflow. It supports atomic in-memory transactions and documents time-complexity for all data operations in O-/Landau-Notation. The persistence is realized by a transactional Append of File (AOF) [43] journal and an asynchronous Copy-on-Write-Snapshot-mechanism [44] that is configurable at runtime. Redis enables up to 250,000 transactions per core and second on custom hardware. In addition Redis can scale up and clustered by an Amazon Dynamo [45]/ Apache Cassandra [46] like hash routing mechanism as used inside the Amazon cloud or at the Facebook data backend.

To realize callback process chaining, non-blocking I/O and good key-value object management we implement the backend logic by use of the Node.js [47] application framework. It is designed to handle functions as first class objects (which is needed for callbacks) and asynchronous event processing by the use of an event loop architecture. Based on Google V8 [48] compiler the Node.js runtime environment is used in critical infrastructures of major software developers like Google, Microsoft and HP/Palm, of major Cloud providers like Yahoo, Heroko and Joyent or financial service providers like BörseGo and Bitfloor. Node.js handles up to 40,000 requests per core and second and can be upscaled by load balancing. Relying on the Google V8 compiler high performance support for Regular Expressions [49] (needed for translations and formatting) and JSON support is given. The key-value nature of AMM CRUD objects is handled by JSON objects [39]. JSON supports lightweight key-value objects that can be schemaless created, modified and persisted. For more efficiency and less understandability JSON can be directly translated to the binary MessagePack format. For finance domain specific cases JSON can also be directly translated to the FIX message format. To keep the AMM domain-independent and universal FIX is optional, but FIX field keys can be realized as reserved key words of the AMM.

The AMM is mostly designed as a scalable and automated trading backend system. Additionally the architecture enables to foster also some innovations for frontend clients and UIs. The stackable API principle allows the end-user to decide which function he wants to use for data management, trading process, semantic, format and session handling. That is why we implement a proof-of-concept Frontend UI prototype that allows managing and combining CRUD objects and callback function. This frontend is realized with the Backbone MVC [50] framework and WebSocket real-time communication.

5. Evaluation, Discussion and Outlook

In this section we follow the DSR process by evaluating the developed design with the kernel theories (Computability, Complexity, Learning Theory and Constructivism) from section 2.3. In addition we endorse that a second evaluation should be performed by interviewing domain experts, which would not be required by the DSR process. As this paper has page limitation, we will shortly elaborate on this in the discussion part at the end of this section, but shift the full evaluation to future research publications.

5.1 Evaluation

The DSR evaluation is structured by checking the AMM implementation (Section 4) against the requirements (Section 3) by use of the kernel theories (Section 2.3):

**Data agility:** The requirement enforces non-fixed or static data structures neither for users nor for scopes.
Trading requirements shall be user-defined. Using JSON allows dynamic (re-)definition of all properties of data objects and collections of objects (bundles) in the AMM implementation [39]. JSON is evaluated as universal as it follows the UNIX principle of text streams [25]. Reserved keywords and permissions are just used where the reliability, the fairness and the trust into the system enforces it for all users. Users can also use domain-specific data structures and formats that can be auto-formatted and semantically converted by the "format" and the "semantic" API. That allows users to define agile and free data structures, rules, priorities and preference at all data scopes to their specific need and rank, match, swap and log depending on this data definitions.

Data Efficiency: This requirement enforces just to define data structures that the user needs for himself, his group or his system. The AMM can be used without any data definitions just for swapping things without attributes. So the user decides how efficient his data structures are. If JSON [39] data structures are not memory-efficient enough, it can be translated to compressed MessagePack structures that are highly efficient and fast data persisting format [40]. As the system requires no data definition (except for the definitions that the user needs for his domain) and as data can be stored highly efficient and fast the AMM follows the principle of data efficiency.

Data Understandability: JSON is a lightweight, human-readable format [39] that can be auto-translated to domain-specific formats like FIX. It allows construction of data objects and attributes by a gentle slope of complexity and gives the user the freedom to define data objects and properties within 5 scopes. In general we evaluate the AMM as understandable by its data structures.

Functional Agility: The requirements enforce a universal and complete set of atomic functions that are extendable by the user. The AMM supports all major characteristics of the Montreal Taxonomy [9]. The UNIX principle of orthogonality [18] is given by the set of functions applicable to user-defined data structures. Callback functions can be dynamically combined to user-defined processes. Users can define additional APIs at client and server side. The AMM application server is implemented in the only mainstream type-free and Lambda-complete language [51]. Type-free, lambda languages are proven to be always Turing-complete [16]. So the AMM is proven to be functional agile.

Functional Efficiency: Requirements enforce atomic, best of breed operations and lightweight designs with thin interfaces. The AMM data store [42] and application server [47] implement atomic operations with a non-blocking event loop. The AMM data store guarantees persistence and data operations that are runtime optimal algorithms: O(1) for swaps, O(n) for n-property CRUD-Objects, and O(N) + O(M*log(M)) for Union and O(N*K) + O(M*log(M)) for Intersection [42]. Memory limit can be overcome by clustering data store and application server instances with load balancing. That proves that the AMM can be implemented with functional efficiency independent from the user-defined structures.

Functional Understandability: Requirements enforce UNIX philosophy and a limited number of simple functions [25]. Each operation in the AMM has one well-defined function. CRUD, rank, match, swap and log plus timing and feedback are 7±2 Miller-aware [30]. So the AMM is functional understandable.

Architectural Agility: Requirements enforce to provide just parts and layers that the user needs. Orthogononality [24] and callback processes [51] support a user selection of stackable and user-defined APIs at runtime on client and server side only restricted by the user data permissions. So the AMM can be seen as architectural agile.

Architectural Efficiency: Requirements stipulate to rely on simplistic data and functional concept. The AMM is proven to be data and functional efficient depending on the users' definitions and a stackable selection of APIs. The level of architectural efficiency is therefore mostly defined by the user’s definition and use of the system and less by the design of the system. Therefore the architecture can be seen as efficient.

Architectural Understandability: Requirements enforce human understandable architecture. Basic APIs (CRUD, callbacks, market models, communication and user) plus semantics and format are 7±2 Miller-aware [30] and thereby enable a gentle slope of complexity [25]. The applications server is programmed in a Smalltalk [23] based language [47] that was designed to be understandable even for children of all ages [52], so it should be also for domain experts. Node.js callback functions follow the principle of glueable code blocks in Smalltalk and of stackable APIs. The API and the Architecture is completely documented on a double paged sheet of paper. So the AMM can be said to be architectural understandable.

5.2 Discussion/Limitations

Besides providing the ability to implement “standard” market models, the market platform also aims at filling gaps in existing research. In its most basic form the AMM provides the basic tools to create market models, processes and services from scratch. In contrast to prior market platform implementations, the AMM is not bound to pre-defined components of market models than can be composed to create a
market setup. Instead, the AMM provides the all-embracing ability to users to define individual solutions to their heterogeneous transaction requirements. Thereby, users are enabled to evolve innovative market structures [3]. With its agile design the AMM is domain-independent, while prior platform implementations have typically been based in securities trading or private auctions. The AMM however can basically be applied to any domain, as the domain-specific characteristics are introduced by the users rather than being pre-defined by the platform. This flexibility is ensured by the fact that end-users can define data rules, priorities and preferences on their own. To further foster this agility, design decisions regarding technology, interfaces, scalability emphasize the pervasive mass-customizability of the platform.

As the proposed the AMM is a meta-system architecture, evaluation can only be conducted on a qualitative level, rather than on an empirical level. Domain-experts that are involved in the use, design and operation of real-world market systems and architectures will to be addressed for empirical evaluation. The holistic AMM approach will be narrowed down to the specific context and use cases of the particular domain expert to conduct such an evaluation. It is planned as next steps to perform evaluations with experts from various domains. In such contextual evaluations the fit of characteristics delivered by the AMM and the actual requirements by domain-experts with respect to higher quality (agility), speed (efficiency) and low costs (understandability) need to be validated.

5.3 Outlook

AMM provides a toolbox in which end-users/domain-experts can conceptualize and create innovative market setups to allocate goods by defining their individual preferences, ranking priorities and matching rules. If user-defined setups prove to be “successful” and are adopted by larger user groups, AMM can provide the setup as a “standard” configuration to the community of users. On the basis of such standardized configurations further on top functionality/services can evolve, creating a new ecosystem of allocation services.

So end-users can invent and create innovation niches specifically to their needs. They can design and establish market models on these innovations, so that market designers can adopt them as state of the art standards. If infrastructures and innovation can foster each other in this way then market innovations become more agile and democratized [3]. Special thanks to Markus Gsell for all the great discussions, his valuable input and the critical reflection on agile market models and to the E-finance Lab at the Goethe University in Frankfurt for supporting this research.

6. References


[50] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, “Design Patterns: Elements of Reusable Object-Oriented Software”, Addison-Wesley, USA, 1994.
