Internal Report 2011-03 February 2011

Universiteit Leiden

Opleiding Informatica

Investigation into a Domain-Specific Language

for Securities Trading Operations

A.C.L. Hunsucker

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Table of Contents

Chapter 1: Introduction

This paper will explore the feasibility of developing a Domain-Specific Language ("DSL"), based on the FIX Protocol, for modeling electronic securities transaction communications. In the past decade, the FIX protocol has emerged as the dominant technical standard for electronic securities trading [3]. As has been widely argued [1], standardization in financial markets offers many benefits in terms of increased connectivity, improved communication, lower transaction costs and lower barriers to entry (improving competition), bringing down trading costs to all parties involved. In fact, these reasons were among those cited for the development of the FIX Protocol (as well as electronic trading in general), and have been proved by its rapid adaptation in financial markets the world over.

However, FIX is hard to use directly by its ultimate users, financial experts – it was designed as a communication standard, not a full-fledged modeling language. As such, it can be expected to be overly detailed in some (e.g. technical) aspects, while falling short in others. Furthermore, being designed with extensibility in mind [2], it is likely to lack definitions strict enough for domain specification. We believe that standardization would be further aided by addressing these shortcomings by developing a Domain-Specific Language for electronic securities trading. Such a language would enable financial experts to implement and design electronic securities communication software without intricate knowledge of the FIX protocol. This paper is intended as a first foray into the development of such a DSL.

In this analysis, we shall restrict ourselves to the domain FIX was initially designed to operate in [Euronext]: trading of equity securities ("stocks") between domestic partners: investment managers (e.g. mutual funds), brokers and trading platforms (e.g. exchanges or "dealer markets"). Specifically, we will limit ourselves to modeling "pre-trade" and "trade" communications (leaving out the "post-trade" phase), consisting of Indications of Interest ("IOI's"), quotes and various forms of orders. Administrative and Session-related messages, while not constituting the focus of our analysis, will also be addressed when material to our model. We will not model cross-border trades, debt/foreign exchange/derivative instruments, allocations, program ("algorithmic") trading or regulatory communications; we will leave these aspects to future research.

Furthermore, we will limit the depth of our analysis, as well as the breadth. For one, along the lines of the FIX protocol, we will limit our exact specification to the communication between parties, leaving the exact semantics of the operations signified by this communication to further analysis or implementation. As will be demonstrated, the benefits of inter-party communication envelop those of in-house processing, mandating the prominence of standardizing communication over operations. Furthermore, the DSD editor we will develop will not be able to generate software, merely allowing its users to design models subject to domain constraints.

Section 2 will start with a history of the electronic trading environment, introducing key concepts of the FIX protocol and domain-specific modeling, as well as providing a rationale for standardization. In Section 3, we will describe the concepts pertaining to our domain and introduce their semantics in our domain-specific language. Subsequently, Section 4 will give

an overview of our metamodel. Section 5 concludes with the editor and example models derived from industry examples. Section 6 wraps up with a conclusion and a suggestion for future research. In the Appendix, we provide an outline of a text-based DSL for describing FIX transactions.

Chapter 2: Background

In this chapter, we aim to describe the broad context of our analysis. We start by providing an abridged history of the electronic trading environment and the electronic handling of securities. We continue by describing the FIX protocol, and follow up with its beneficial effects on the securities industry. Finally, the process of domain-specific modeling is described.

2.1 The Genesis of Electronic Trading

Since the dawn of securities trading, financial instruments such as shares and bonds have been held in the form of certificates. Usually bulky and ornate pieces of paper, these documents physically represented ownership of a share [4], and ownership was transferred by transferring the certificate, along with the processing of significant volumes of paperwork [5]. Stockbrokers – intermediaries between end investors and the exchange – would employ "runners" to physically carry the securities between firms, on handcarts or in large wallets [4].

In the post-war years, as personal wealth increased and the memory of the Great Depression faded [4], ownership of shares by the general public ballooned; in 1960, approximately three million shares were traded daily on the New York Stock Exchange, six years later this had increased to 13 million [6]. This increase precipitated the "paper crunch", when broker firms were unable to physically process securities transactions and dividend payments in time. In April 1968, the backlog in NYSE-listed securities processing amounted an estimated $2.67 billion [5]. The physical certificate system became unworkable, and a solution was mandated. Generally, two (non-complimentary) remedies for this problem exist: *dematerialization* and *immobilization*.

Dematerialization entails the severing of the link between the share and its physical representation (the certificate) – in other words, representing shares purely in digital form [7]. In this way, transfer of ownership can be effected merely by means of an electronic transaction. Immobilization, on the other hand, maintains the need for certificates, but negates the need for physical movement by "immobilizing" the securities in a Central Securities Depository. Ownership is then transferred by means of (electronic) bookkeeping [7]. The latter method had been used by the *Vienna Giro Association* since the 1870's [6]. In 1973, the Depository Trust Corporation was set up to this end in the United States, and in 1975 immobilization was made mandatory by the Securities and Exchange Commission [7].

The developments described above mainly pertain to the act commonly described as "settlement", which is a post-trade process where ownership changes are effected. Shortly afterwards, electronic trading of equity securities commenced. Arguably, this would not have happened had securities still existed exclusively in their physical form.

The NASDAQ, an "over the counter" or OTC market, was the first instance of large-scale electronic trading from 1971 onwards [8]. However, exchanges (as opposed to OTC markets) remained dependent on human interaction. Trading was mostly conducted by telephone, often in conjunction with "open outcry" systems where traders would communicate directly

(verbally or by a complex sign language) in so-called “pits”. The idea of a “fully automated exchange”, where order books are processed automatically and ownership of (dematerialized) securities is recorded and changed automatically, was first proposed by Fisher Black [9] in 1971. The first actual transition to electronic exchange trading was Toronto’s TSE, implementing the first true ECN (Electronic Communications Network) trading system in 1977 with various major exchanges following suit throughout the 1980’s and 1990’s. [10] While slower on the uptake, bond markets began the transition in 1999, reaching near-exclusive electronic trading by 2006 [8]. Domowitz provides a very extensive overview of electronic trading systems with their years of establishment in [11].

Electronic trading has numerous advantages over “traditional” trading through human interaction. For one, less or no “rekeying” (the manual entry of order data) is needed by the intermediary, reducing transaction cost and potential for erroneous processing [1, 12]. The logical end of this transition is the possibility of “Straight Through Processing”, the seamless integration of the different parts of the trading process by linking the execution, confirmation, clearing and settlement phases of the trade with risk management procedures [12], without the need for re-keying or even human intervention. In this way, electronic trading moves beyond front-office operations into involving (and streamlining) back- and middle-office operations as well. Furthermore, risk management is made more comprehensive and errors in processing less likely. Also, scalability is greatly enhanced – while it is very expensive to expand a trade floor, it is trivial by comparison to employ additional computers and bandwidth [12].

Numerous downsides are also noted in literature. First, there is the issue of counterparty risk – the risk of the other party in a deal failing to meet its obligations. Traditional trading mitigates this risk by putting parties in direct contact, engendering mutual trust. Electronic trading systems provide anonymity and impede verbal communication, making it harder to ascertain the counterparty’s identity and trustworthiness. [12] cites this as an important factor in the slower uptake of electronic trading in the foreign exchange market, where counterparty risk is greater. Furthermore, the danger of reduced liquidity is mentioned in [12] and [13], either due to a reduction in brokers (through increased competition) or an absence of market-makers and increased fragmentation. Thus far, such fears have proven baseless [12].

2.2 The FIX Protocol at a glance

The emergence of electronic trading called for a standard messaging protocol. Initially, exchanges or platforms developed their own standards to connect buyers and intermediaries to each other and the market. This led to high costs of establishing connectivity, prohibitive switching cost, impediments to competition and unnecessarily high trading cost [1]. In 1992, Salomon Brothers (an investment bank) and Fidelity Investments (a mutual fund) decided to develop a communications standard for electronic trading of equity securities ("shares"); this would become the FIX protocol. In 1995 the FIX committee, by then the driving force behind FIX, released FIX 2.7, the first production version of the protocol [14]. Since then, FIX has been maintained as an open standard, around which interested parties may develop commercial or open source software [15]. The FIX protocol is directed by FIX Protocol Limited, or FPL, and its Global Committee [16]

During the past 15 years, FIX has established itself as the *de facto* securities trading protocol [3]. While exact usage figures are hard to come by [15], a 2008 survey indicated that 75% of buy-side and 80% of sell-side firms, as well as three quarters of all exchanges, use FIX for electronic trading. The FIX protocol is constantly expanding its scope, having started as a protocol for bi-lateral domestic equity trading, it has now evolved to support post-trade processing, transaction handling on behalf of third parties, and a variety of instruments like derivatives and foreign exchange (currencies) in a cross-border context.

It is important to note that FIX itself does not imply or endorse any specific implementation. FIX is not software [15], it is an open and free communications standard around which interested parties (members and non-members) can develop their own implementation. Furthermore, an attempt has been made to steer clear of "over-standardization" [2]: the protocol does not define a standard for the underlying network (e.g. leased line, internet, satellite) or network protocol (e.g. TCP/IP). Likewise, support for encryption is present, but the choice of encryption method is left to the parties involved in designing the connection [2]. Interested readers are referred to [3] for a sample implementation.

The FIX protocol defines a *message* as a ASCII-encoded string, consisting of a standard header, a "message body" and a standard trailer [2]. Messages consist of *<tag = value>* fields of an undefined sequence (barring a limited set of exceptions), with the *tag* consisting of a predefined number and the *value* in alphanumerics. The standard leaves a great amount of tag values open for proprietary extensions, allowing implementing parties to embed their own features into their implementation of the protocol. This extensibility even allows for "vanilla" FIX features being "carried back" into older versions, that initially did not support them [17]. More importantly, there is no central record of which tags have been used for proprietary extensions, aggravating inter-vendor compatibility and possibly endangering the benefits of interoperability as described in section 2.3.

The protocol defines three message groups: *session messages*, *administrative messages* and *application messages* [2]. The first two pertain to establishing and maintaining a connection: they define logon/logout messages, a framework for authentication (left to the specific implementation), and two distinct mechanisms for maintaining session integrity: the *heartbeat* (periodic messages sent by all parties) and a message sequence number allowing

for discrete gap recovery. These messages are not considered fundamental to our analysis, and will hence not be detailed further; we will focus (mainly) on the third type, application messages, defined as “the exchange of business-related information” [2]. Elements of FIX bearing relevance to our analysis are detailed in the next section.

The abovementioned practice of providing freedom of implementation, as well as the extensible nature of the FIX tag system, gives rise to a host of proprietary implementations that are mutually incompatible and may provide non-overlapping functionality. As a result, parties are likely to use a specific subset (or superset) of the FIX protocol. Buy-side institutions must therefore ensure that their Order Management Systems are compatible with the protocol implementation used by their sell-side institution, and ultimately, the trading platform or exchange used. We will elaborate further on this condition in Section 4.

It should be noted that from FIX 4.2 onwards, FIXML is being developed concurrently as an XML vocabulary based on the FIX Protocol [16]. In other words, it defines a one-to-one mapping from FIX’s conventional tag=value form to XML. It is intended to utilize existing systems and processes and protect investment in “traditional” FIX implementations, while providing a migration path to next-generation FIX systems and increase interoperability with other industry standards. Furthermore, FIXML does not define a session or transport protocol, relying on traditional FIX for embedding or requiring the use of a different transport protocol [17]. We will not cover FIXML in our analysis.

2.3 The Benefits of Standardization and the FIX protocol

The benefits of implementing FIX are described extensively in [1]. They mirror in large part those of standardization in general, with some added gains applying to the financial context in which the FIX protocol is used. We will briefly reproduce the key gains in this section.

First, it should be noted that the automation of trade systems started *within* organizations. While using a standardized protocol for internal operations may benefit firm-wide efficiency, applying standardization to external (i.e. inter-firm) links engenders a multitude of additional benefits. The most direct effects of widespread inter-firm FIX use are a reduction in connectivity costs and an increase in efficiency level.

Connectivity costs are lowered when a single standard is used between all parties. This way, the level of investment needed to establish trading connections decreases rapidly as the number of connections established rises. With a single connectivity standard, no new software needs to be developed and with it, less individuals trained and less testing and certification is called for. This, in turn, allows all market parties to achieve a more optimal level of domestic and international connectivity. For instance, institutional investors are no longer confined to a single brokerage firm for their market operations, and brokerage firms may more easily interact with numerous trading platforms, allowing them to offer a more comprehensive array of investment services to their clients. Trading platforms, in turn, may more rapidly achieve a number of trading members deemed sufficient for proper functioning.

This increase of competition benefits the whole securities marketplace. When switching costs go down, investors are more likely to seek out the brokerage firm that offers its services at the most attractive price level. Together with the decrease in connectivity costs as a whole, this is likely to significantly reduce transaction costs in the marketplace, bringing about greater efficiency. This in turn affects the overall level of activity and liquidity in the market, reducing bid-ask spreads for end investors. This process is furthered even more by increased global market integration made possible by adaption of a standardized protocol. All in all, higher net returns and a more optimal asset allocation greatly benefit end investors, and with them, large sectors of the global economy as a whole.

However, there are a few likely impediments to this transition to a standardized electronic securities market. First of these is the free-rider problem: while the benefits of standardization described above arguably accrue to a wide range of market parties, the costs are borne by a select few. Second, the standard may not cater fully to the needs of all parties, giving rise to proprietary and potentially conflicting additions. Plus, the increase in competition as described above erodes the market power of individual participants, who may thus oppose a standard that is thus detrimental to their profits. Finally, the network effects of an extant standard may be too strong, in that they hinder innovation and impede implementation of superior standards.

FIX Protocol Limited seeks to remove these hindrances, and aid implementation, through its organization as a non-profit membership organization. In this way, the free-rider problem is alleviated, the organization itself has no market power and market participants get an equal say in its decisions.

2.4 Domain-Specific Languages and Modeling

The use of domain-specific knowledge is increasingly gaining currency in systems analysis and development. Literature often (but not always) distinguishes *domain-specific languages* and *domain-specific models*. The former are defined as “a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain [18]. Mernik and Heering state in [19] that domain-specific languages exist “in a gradual scale with very specialized DSLs such as BNF (*Backus-Naur Form*) on the left and GPLs such as C++ on the right”, adding that “domain-specificity is a matter of degree”. This vagueness notwithstanding, DSLs as a whole offer strong advantages. Gray and Karsai identify several key benefits in [20]: conciseness, a boost in productivity, and the ability “of a computationally naïve user to describe problems using natural terms and concepts of a domain with informality, imprecision, and omission of details”, to which Latry et al add the admittedly large benefit of direct verification by means of domain restrictions as specified in the DSL [21]. Van Deursen et al mention “improved readability, maintainability, flexibility and portability” as benefits of Domain-Specific Language use in [18].

Whereas Domain-Specific Languages are usually textual [22], Domain-Specific Models usually represent domain concepts in a graphical fashion [23]. Please note that this

distinction has become blurred, with “domain-specific language” also referring to “visual programming languages” (metamodel editors) as well as the more conventional textual incarnations. DSM use offers similar benefits, such as a 5-10x productivity increase and a drastic increase in reusability [24]. DSMs usually specify a *solution domain* as well as the conventional *problem domain* – the latter is comparable to the “specific application domain” used in the definition by Mernik and Heering as cited above; whereas the former refers to a (textual) language that a DSM “program” is translated into. Literature mentions general programming languages as solution domains, such as C++, assembly language and XML [25] but also domain-specific languages such as SQL [ibid], CPL [22] or a textual DSL developed concomitantly with a visual DSL/DSM in a two-pronged approach [21].

Domain-specific modeling or language development has been applied to a diverse variety of problem domains, such as machine control and call processing [22], graphics and networking [ibid], server configuration and medical purposes [25]. A few instances of problem domains in the area of finance are also known, such as the RISLA language for interest-bearing financial products [26] and a language for the definition and processing of financial contracts, such as swaps and options [27]. We are aware of no language or model specifically developed for transactions of such financial products. Given the definition of a DSL cited above, together with the notion that domain-specificity is a continuum, it can be argued that FIX already constitutes a domain-specific language for describing transactions in various financial products (i.e. securities). However, FIX was never intended as such: its domain is not strictly defined, it is burdened with concepts not pertaining to the domain (such as session and administrative messages) and its notation, using numerical tags, hardly lends itself to be “manipulated by a domain expert to express a solution as a model” [21]. This paper attempts to remedy this deficiency by developing a domain-specific language that largely mimics or incorporates FIX’s functionality, but is more expressive, intuitive and strictly defined.

Chapter 3: Domain Definition

This section will introduce the key concepts to be incorporated into our domain. These concepts are based on abstractions derived from the FIX language; we will give a brief description of the FIX underpinnings.

In this section, we use the nomenclature of [28]: 'DSL' or 'Domain-Specific Language' refers to the language we intend to develop, 'DSD' or 'Domain-Specific Definition' is a "program" written in such a DSL and the 'DSP' or 'Domain-Specific Processor' is "a software tool for compiling, interpreting or analysing domain-specific descriptions".

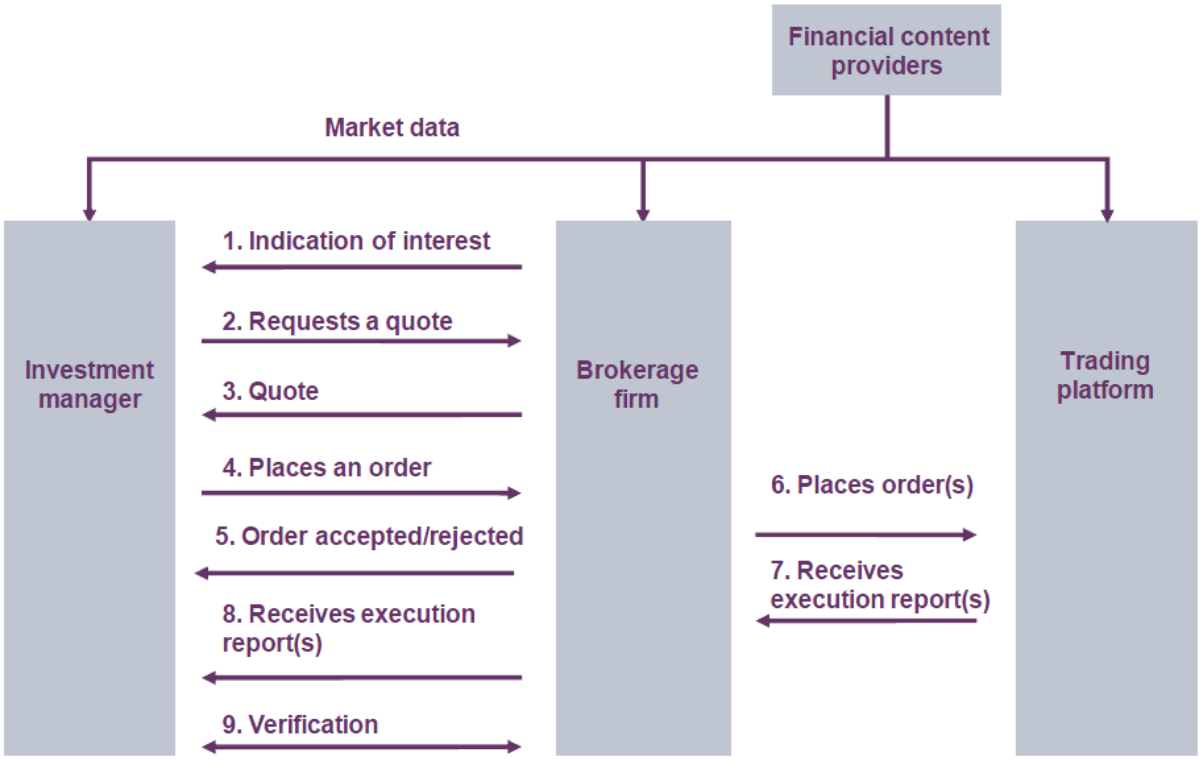
3.1 Parties

By definition, a trade implies at least two parties involved in a transaction. FIX communication, at its essence, usually takes place between two parties. For message exchange to take place, a *session* must be established [2]. This entails two distinct stages: establishing a physical and logical connection between two endpoints (left to the implementation), and the subsequent establishment of a FIX Session. As defined in [3], "A FIX session is defined as a bi-directional stream of ordered messages between two parties within a continuous sequence number series"[3] . Sessions are the overarching "meta-state" of the system: they define the context in which message exchange take place. Not only do they allow for authentication of sender and receiver ("initiator" and "acceptor" in FIX parlance), they also define the identity of these parties: while FIX messages have address fields in the header, FIX lacks, unlike a protocol such as TCP/IP, a well-defined system for name resolution (e.g. DNS), routing, or even a system of verifying whether these names are correct or unique. The FIX specification only mentions the existence of the fields, leaving much to the implementation [2]. In version 4.0 of the protocol, these fields were extended to allow for delivery of messages "on behalf of" third parties [29]. In version 5.0 of the protocol, the session and application message groups were decoupled further with the introduction of the *Transport Independence Framework*; from this version on, FIX application messages can be exchanged using a multitude of transport technologies, including the "traditional" FIX session [30]

It should be noted that the strictly two-party architecture of the session layer in no way implies that FIX may only be used for transactions between the two parties initiating the session. On the contrary, the added value of a standardised interconnected electronic trading system lies in large part in its ability to foster connections in a "many-to-many" fashion (as detailed in section 2.3). As a result, the situations in which one is likely to encounter FIX "in the wild" are likely to involve a multitude of parties. In such an environment, multiple sessions exist concurrently. Take for instance the example given in [14]: a client contacts his broker with a "sell" order, upon which the broker sends Indication of Interest messages to several of his clients, one of whose places a "buy" order with the broker. In the end, the seller sells securities from his portfolio to a seller, through their broker. This is an example of a "meaningful" FIX transaction involving multiple parties and multiple sessions: between the seller and the broker, and between the broker and its clients (including the eventual buyer). Furthermore, the delivery and executing of messages "on

behalf of" third parties, as made possible in FIX 4.0, also provides a mechanism for involving more than two parties in a single session [29]

As mentioned above, the *initiator* and the *acceptor* partake in a FIX session and, in general, constitute the parties in the underlying securities transaction (barring "third party" transactions). We will use *parties* to refer to the union of initiators and acceptors. More specifically, in this analysis we will focus on *investment managers*, *brokerage firms*, and *trading platforms*, as identified in [1]. These three will be the main entities in our DSL.



Source: Oxera.

Figure 1: generalized interaction between parties

3.1.1 Investment managers

Investment managers, "buy-side institutions" or "buy-side firms" are professionals managing securities portfolios on behalf of their clients or principals, which can be (wealthy) individuals, pension funds, insurance companies, mutual funds, ETFs ("exchange trade funds") or any other "firms which trade securities to make a return on an initial capital investment [1]. They are the ultimate owners of the securities and other financial instruments traded in a market, and the overwhelming majority of trades is performed on their behalf. It should be noted that the phrase "buy-side" does not imply that these firms are only involved in purchasing securities – rather, they may sell as well as buy securities from their portfolios; the "buy" refers to the purchasing of market services, not to the actual securities bought. These services entail access to a *market* or *trading platform*, which is

usually accomplished through a broker, whose services are purchased. In our DSL, investment managers are represented as a generic BuySide entity. The DSL allows users to name these entities. Buy-side entities may only be connected to Sell-side Entities (Brokers), by means of *interactions*. Typically, buy-side entities will have at least one connection with a sell-side entity. We will describe these interactions in section 3.3.

3.1.2 Brokerage firms

Brokerage firms, also called "brokers" or "sell-side firms", are defined in [1] as "firms that provide execution service to investors". In short, they provide "market access" to investment managers – and by extension, to the clients and principals they represent – allowing them to trade securities. This process is greatly aided by the dematerialized and/or immobilized nature of their portfolios, as discussed in the previous section, and by the electronic transmission of orders. They are called "sell-side firms" because they sell market access services to investors. These same firms may also trade and own securities on their own behalf and for their own profit, or on behalf of other business units within the same firm. In these cases, there is no other ultimate beneficiary than the firm itself. The FIX protocol defines a special tag to distinguish these trades. Please note that a connection to a trading platform is not necessary for a broker to function – not all transactions need one. For instance, the example in [14] displays a broker-mediated transaction between two of the broker's clients, which does not involve a trading platform. Naturally, this transaction may still involve a third party in the post-trade *settlement* phase, however, this is not part of our analysis.

In our DSL, brokers are represented by a generic SellSide entity, with a user-provided name. Brokers may have bidirectional connections with buy-side entities as well as trading platforms, though not with each other. Each sell-side entity is assumed to be connected to at least one buy-side entity, and possibly (though not required) one trading platform. Please note that this requirement is not enforced – brokers may also trade on their own account (negating the need of buy-side clients) and deals do not necessarily involve a trading platform [14].

3.1.3 Trading Platforms

Trading platforms, or *markets*, are defined as (stock) exchanges, ATSs (Alternative Trading Systems), MTFs (Multilateral Trading Facilities) and Crossing Networks [1]; in short, they are a venue for matching buyers and sellers. Stock exchanges, the most well-known form of trading platforms, are regulated entities that allow trading in securities and instruments that have obtained a listing on that specific exchange. While stock exchanges have a natural predisposition towards listing stock of companies originating in the exchange's country, major exchanges radiate beyond natural boundaries, and major international corporations often list their stock on multiple exchanges. A host of major exchanges offer FIX connectivity, usually as part of a larger offering of connectivity solutions, and bound to an exchange-specific implementation. An example is given by the Euronext exchange network in [3], and a list of participating exchanges is provided by FPL at [15].

Alternative Trading Systems are regulated US-based non-exchange trading venues that, like stock exchanges, “constitutes, maintains, or provides a market place or facilities for bringing together purchasers and sellers of securities or for otherwise performing with respect to securities the functions commonly performed by a stock exchange” [31]. They do not “list” securities in the fashion stock exchanges do, even though they may trade securities that are listed on an exchange. Trading in this fashion is usually called “OTC”, for “over-the-counter” trading to distinguish it from trading on an exchange. Most ATSs offer FIX connectivity, using proprietary implementation. An example of such is given in [32]. The Securities and Exchange Commission provides a list of Alternative Trading Systems in [33].

In a similar fashion, Multilateral Trading Facilities are defined as “a multilateral system, operated by an investment firm or a market operator, which brings together multiple third-party buying and selling interests in financial instruments” [34]. MTFs serve as a transnational, low-cost alternative to exchange-based trading, and as such compete with one another. Examples of MTFs are Chi-X Europe, Turquoise and Burgundy. Most, if not all MTFs offer connectivity based on their respective FIX implementations.

Crossing Networks, finally, are systems that “cross” or match buy and sell orders from connected members in an automatic, “black-box” manner. Unlike ATSs, they usually operate in an anonymous fashion. Examples of Crossing Networks are Goldman Sachs’ SIGMA X, Nomura’s NX (also operating as an MTF within the EU), Liquidnet and Pipeline. All these systems offer FIX connectivity in line with their proprietary specifications. An example of such is given in [35].

In our DSL, trading platforms are specifically named, that is, they are not generic. This stems from the prevailing differences in connectivity requirements among trading platforms, as discussed in the previous chapter. As such, it is non-trivial for a broker to establish a connection to a certain trading platform, as this will entail additional effort and investment in the FIX implementation. Given the multitude of FIX-based trading platforms, we provide a non-exhaustive list, aiming not to cover the entire spectrum. Extending the DSL to include more named platforms is relatively easy.

Brokers may connect to trading platforms, but not every interaction is valid. For instance, quote messages are invalid in this context as price propagation is assumed not to use the FIX protocol. IOI’s are equally irrelevant in the context of trading platforms. Consequently, we only allow Ordering and Execution Reports to be exchanged between brokers and trading platforms.

3.2 Messages

Message transfer – the sending and receiving of "Application" messages between the session initiator and acceptor – is what gives the FIX protocol its functionality. Transactions (the buying and selling of securities) take place after the relevant messages have been sent and processed; the FIX messages are therefore highly relevant to our domain. As such, the "arcs" or connections in our visual DSL will be "interactions", which are aggregates of FIX messages defined later in this section.

Messages consist of *<tag>=<value>* fields delimited with the ASCII 01 ("start of heading") character. Tags are numbers that denote the type of data in the field, such as the name of the sender or the stock's symbol. Messages start with a standard header, itself made up of *<tag>=<value>* fields, and end with a trailer. There is a taxonomy of *message types* (denoted by a field in the header), each serving a particular purpose: messages for ordering, reporting, and soliciting orders, but also for logging in and testing the connection. Each message type requires a specific set of tag fields (for instance, a login message doesn't need the field for "currency type" but it needs a field specifying the type of encryption desired for the session), which are specified in the FIX documentation. It is important that these fields all be present, however, the order in which they are sent is usually not important, unless otherwise stated.

Messages can also refer to previous messages – for instance, a *Quote Request* message may be responded to by a *Quote* message from the other party, containing the same value in the *QuoteReqID* field. Also, a *New Order* message may be cancelled using a *Cancel Request* that refers to the order message to be cancelled, using its OrderID. FIX does not provide or define a system to generate these IDs, leaving it to the sender's implementation.

The message types relevant to our domain will be covered next. Please note that this overview is based on version 4.0 of the FIX Protocol as described in [2].

3.2.1 Indications of Interest

IOI's, or *Indications of Interest*, are messages sent by the brokerage firm indicating that it wishes to buy or sell a specific security. The broker may do so in an agency capacity (on behalf of a client) or proprietary (for its own gain), this is indicated by the optional *IOINaturalFlag* tag in the message. Other important fields specify *Side* (whether the interest is in buying or selling), the *Price* (optional), expiration time for the offer (optional), the security name and the amount of securities to be bought/sold. IOI's are usually sent to a large number of clients, which may then respond by ordering a buy/sell from the broker, who fills one or more orders, until the desired number of securities is bought/sold.

3.2.2 Quotes and Quote Requests

Quote Requests are sent to brokers, to request a price quote from them for buying or selling a specific security. The broker usually responds with a *Quote* message. Note that Quotes and Quote Request differ from IOIs, in that quotes are for a specific security and quote requests

are sent *to* brokers. Quotes may also be sent unsolicited, i.e. without a preceding quote request message. Solicited quotes need to include the appropriate value in the *QuoteReqID* field, whereas unsolicited quotes lack these fields. A Quote Request message only requires the security's symbol, and optionally the Side (buy or sell) and the securities previous closing price (for additional verification). Quote messages also include the bid (and optionally) offer price, and may also include a stated quantity and validity time.

3.2.3 New Order – Single

The *New Order – Single* message is used for actual trading, sent by clients/intermediaries to brokers indicating that they wish to buy or sell a stated security. Mandatory fields include *Side*, quantity, stock symbol, handling instructions (private/public/manual) and order type. This last field allows, apart from "Market" orders (straight buying and selling), for limit orders, stop orders, etcetera. Extra optional fields are provided for these order types, as well as execution instructions.

3.2.4 Execution Reports

Execution Reports are sent by the broker to report on the status of a previously submitted order. They may flag an order as New (received, but not executed), Partially Filled, Filled (completed), Done for Day (filled, no further executions forthcoming), Cancelled, Replaced, Pending Cancel/Replace, Stopped (at the exchange), Rejected (by broker), Suspended (usually at request of client), Pending New, Expired and Calculated (signaling that fees have been calculated and returned in the message). Execution Reports are sent autonomously or in response to an *Order Status Request* message. It contains mandatory fields listing the quantity filled (the amount of securities purchased) and the average price, as well as various status-related optional fields.

3.2.5 Don't Know Trade

Don't Know Trade (DK) messages are sent as a notification that a trade has been rejected. A reason is given in the (mandatory) *DKReason* field, for instance "Unknown Symbol" or "Quantity Exceeds Order". Other fields include the symbol (presumably echoed from the New Order message), side, last price and last quantity.

3.2.6 Order Cancel / Replace Request

The *Order Cancel / Replace Request* message allows clients to change the parameters of an existing order. Only orders that can be successfully pulled back from the exchange floor without executing can be altered, and only a limited number of fields can be changed, such as price, order quantity, execution instructions and the expire time of the order. These fields are all included in the message (though not all mandatory). For identification purposes, this message is treated as a "replacement order" with its own OrderID. When the order cannot be changed, an *Order Cancel Reject* message is returned. Unlike the name suggests, FIX does not recommend the use of this message to cancel outstanding orders.

3.2.7 Order Cancel Request

The *Order Cancel Request* message allows clients to request the cancellation of all or part of the remaining quantity of an existing order (denoted by the mandatory *CxlType* field). Only orders that can be successfully pulled back from the exchange floor without executing can be canceled. A Cancel Request is treated as a separate order for ID purposes, and given a proper *OrderID*. If cancellation fails, an *Order Cancel Reject* message will be generated by the broker.

3.2.8 Order Cancel Reject

Sent only in response to an *Order Cancel / Replace Request* or an *Order Cancel Request* message, denoting that these requests cannot be honored. For instance, an attempt might be made by the requester to modify an order that has been already filled. A reason for rejection can be given in the optional *CxlRejReason* field. A mandatory field is included that refers to the ID of the rejected message.

3.2.9 Order Status Request

An *Order Status Request* message is sent to request an *Execution Report* from the broker. The *ClOrdID* field must contain the ID of the order whose status is being requested. Other mandatory fields are *Symbol* and *Side*.

3.3 Interactions

Interactions are the arcs between entities in our DSL, representing the link between parties. In the FIX protocol, parties interact by exchanging messages as detailed in the previous section; given the large number of possible messages and their fine-grained nature, we have decided not to work directly with messages in this DSL. Instead, FIX messages are aggregated into interactions for the purpose of brevity. For instance, in the customer/broker example in [14], Customer A's order to sell securities consists of three messages between buyer and seller alone: an "Order Single" message sent by the customer, an "Execution Report" sent in response by the broker, followed by a second "Execution Report" to indicate completion. In between, the broker has sent numerous messages to its clients to find a buyer for Customer A's security, Customer B. All in all, the sale of A's security to B requires no less than seven messages. This number grows even larger when we take into account the mandatory session messages, the possibility of resends, and Cancel/Reject/Replace messages that might occur.

Furthermore, there are usually several messages defined for what is globally the same goal. The FIX 5.0 protocol defines ten messages pertaining to Quotes alone [36], and nearly twenty messages related to Orders [37]. This is a dramatic increase from earlier versions of the protocol. We believe that, while this plethora of messages possess (slightly) distinct semantics and implementation characteristics, we may 'abstract away' some of this distinction into more general interactions, such as Quoting, Buying, and Selling. We will now cover these generalized abstractions, based on the messages described in Section 3.2.

3.3.1 Indicating Interest

Indicating Interest is mapped to the *Indication of Interest* message described in 3.2.1. It also includes any IOI's replaced or cancelled using subsequent message with a different *IOITransType* flag. Due to this narrow mapping, the semantics for this interaction are very similar to those described in 3.2.1, and will therefore not be detailed here. The directionality of the arc indicates the origin of the message – not whether the originator is interested in buying or selling. This may be indicated by adding a description to the arc. The arc may also be tagged with a security symbol and quantity. IOI arcs always lead from sell-side to buy-side firms.

3.3.2 Requesting Quotes

Requesting Quotes are sent by the investment manager to the broker to request a price quote on a stated security, as described in Section 3.2.2. This interaction is originated by the buy side (investment managers), and entails Quote Request messages (and their variants defined in version 5.0) sent to the broker as well as the Quote Request Reject response defined in version 5.0. It does not include the actual Quote message interaction requested in response, which is described in section 3.3.3. Please note that requesting a quote in this way is not mandatory and is only used in some markets [2]. The DSL allows the user to specify whether the quote requested is for buying or selling, and which security is involved and in which amount. Arcs always start at Investment Managers and lead to Brokers.

3.3.3 Quoting

Quoting entails the sending of Quote messages (as defined in the FIX 4.0 protocol as well as in section 3.2.2 of this document), its subsequent extensions in version 5.0 (e.g. Mass Quoting) and all variants such as unsolicited quotes, quote rejects and quote status messages. In short, this interaction entails the propagation of price quotes to buy-side from sell-side; Quoting arcs therefore always originate at Brokers and lead to Investment Managers.

3.3.4 Ordering

Ordering interaction entails the buying and selling activities in the greater securities marketplace. It encompasses the Order Single message (and all its variants possible through manipulating the *OrdType* and other fields) as described in section 3.2.3, as well as list orders and the messages described in sections 3.2.6 through 3.2.8. Orders are either sent from investment managers to brokers or from brokers to trading platforms. The interaction arc can be 'tagged' as Buy or Sell, and may also include the security's name or symbol as well as quantity.

3.3.5 Accepting/Rejecting Orders

This interaction consists of a subset of the Execution Report message as detailed in 3.2.4 – namely, those with the *OrdStatus* field set to 0 (*New*, meaning accepted) or 8 (*Rejected*) as well as the Don't Know Trade message as described in Section 3.2.5. Both denote the acceptance or rejection of an order. All other Execution Report messages (e.g. fills) are described in the next interaction. These message are sent from the broker to the investment manager, or from the trading platform to the broker. However, this interaction also encompasses Order Status Request messages (3.2.9) and Execution Report Ack [30] messages sent by the investment manager. The interaction arc can be 'tagged' as Accept or Reject.

3.3.6 Execution Reports

This interaction encompasses all other types of Execution Reports (i.e. those not described in 3.3.5) sent by the broker or trading platform. For example, a "fill" is an Execution Report message, with the *OrdStatus* field set to 2, denoting the successful completion of a buy/sell order. It may also be used to notify of expired orders (when a time limit is given in the original order message) or to confirm the (requested) cancellation of an order. See [2] for a full description.

Chapter 4: Metamodel

This chapter describes the domain-specific model, or metamodel, underlying our domain-specific language. The metamodel was developed based on the semantics described in the previous section. It is presented here in Ecore format, as visualized by the Eclipse Modeling Framework. The entities are formed by the three generalized parties previously described (in the top layer of figure 2) as well as the interactions which constitute the various relationships (in the middle layer). All entities – parties and interactions – have several attributes as described in the previous section.

4.1 Metamodel overview

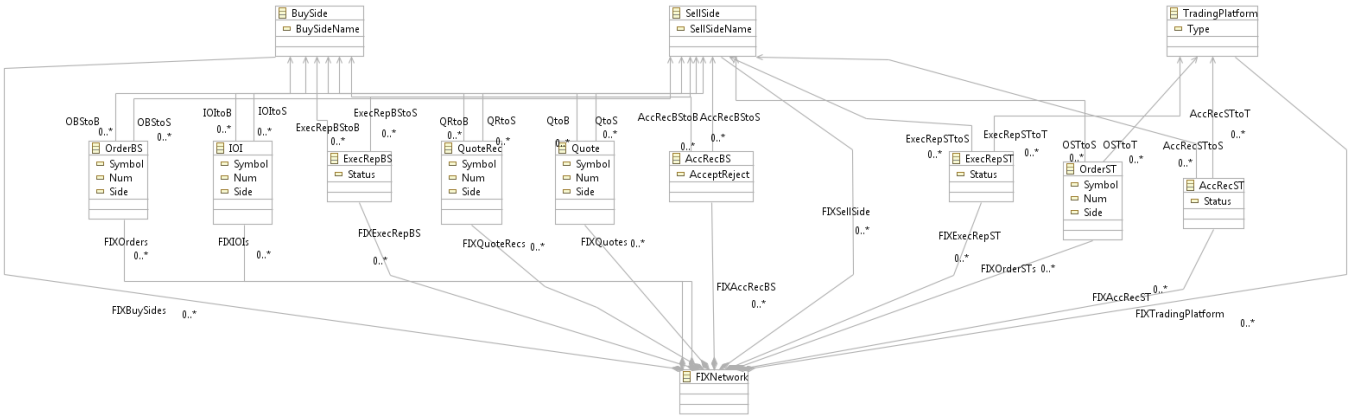


Figure 2: The metamodel

Note the “FIXNetwork”EClass-entity at the bottom. While it has no meaning in our domain, it is mandated by the Eclipse Modeling Framework as a “mother” class representing the model, from which all other entities (classes) inherit. It has “bulbed” arcs leading to every entity in the metamodel, these are called aggregations. Their cardinality specifies the valid number of entities in our model; each aggregation has a cardinality of “0…*” to denote that a model may contain an infinite amount of parties and interactions between them. These arcs are named “FIXBuySides”, “FIXIOIs”, “FIXQuotes” etcetera, after the entities they connect.

The middle layer in Figure 2 contains the EClass-entities representing the interactions. They have several attributes, and have association arcs (with arrowheads) to signify which parties they may connect in a model. These arcs have cardinality, to signify the nature of the relationship. In our metamodel all associations have “0…*”, or zero-to-many, cardinality to signify that a party may have several interactions with several parties. These arcs are named after the entities they connect, abbreviated to reduce clutter. For instance, “QRtoB” connects the Quote Request entity to the Buyside entity, and its brother “QRtoS” connects to the sell-side entity. Some interactions are possible between multiple pairs of parties – for instance, Order interactions may lead from a Buy-side to Sell-side entity, or from a Sell-side entity to a Trading Platform; in these cases, “BS” and “ST”, respectively, are affixed to the interaction name. For instance, whereas there is only one “IOI” entity in the metamodel, the order interaction is represented by “OrderBS”and “OrderST”.

Please note that, in this metamodel, there is no clear visual distinction between the parties and the interactions – all are represented by EClass-entities, whereas in the eventual editor there will be a fundamental difference between parties (entities) and interactions (relationships between those entities). This distinction is made at a later stage of the implementation process, during the mapping model creation phase. Directionality of the relationships is also defined during this process and can hence not be gleaned from this diagram. The association arcs in this metamodel however serve to define one restraint, that of valid interactions: no entity may interact with another of the same type, and interactions are only valid between specific types of parties (e.g. a trading platform does not send out IOIs, and is hence not connected to the IOI entity).

The top layer of the diagram contains the party entities: BuySide (representing Investment Managers and the like), SellSide (brokers and other entities providing market access to buy-side entities) and TradingPlatform for exchanges, MTFs, ATSs etcetera.

4.2 Towards a Domain-Specific Language

The metamodel described above forms the basis of a Domain-Specific Language. For the DSL to be useable by domain experts (financial experts, in our case) a *Domain-Specific Processor* [28] is needed to transform models, developed using the metamodel as instances of the problem domain, into output as instances of the solution domain. This DSP takes a DSD (a "program" specified using the DSL) and applies *evaluation semantics* [27] to achieve this transformation. Ideally, this transformation is made simple as verification is rendered unneccessary by the DSL itself, as described in section 2.4 and in [21]. Nonetheless, for this transformation to be possible, both the DSL syntax and the evalualation semantics need be unambiguous and complete.

Our analysis has stopped short ofproviding full, formally defined evaluation semantics. Nor have we specified a solution domain for our prospective domain-specific language. Nevertheless, using the metamodel developed in section 4.1 and the informal semantics given in section 3, we may develop an editor which allows domain experts to design syntactically valid models. An example of such an editor will be given in Chapter 5. In the remainder of this section, we will informally touch upon possible uses and problem domains for our prospective DSL.

4.2.1 Generalized interaction

With the DSL editor developed for this analysis, it is possible to model *generalized interaction* between market parties in an electronic trading environment. In other words, it allows the user to specify a certain connection arrangement: the design of an electronic trading network. In this way, the parties represent actual parties, but the interaction specified does not all take place at the same time; rather, it shows what transactions *may* take place. Generally, no symbols or quantities are specified, and Side (buying or selling) may also be left empty to indicate that both are possible. Diagrams like this are found in the documentation of specific trading implementations (e.g. [3], [32], [35]) and in [1]. Possible

solution domains for a generalized interaction DSD are configuration files (runtime as well as compile-time), automatically generated documentation or a requirements specification. An example will be given in Section 5.1.

4.2.2 Actual interaction

The DSL may also be used to "log", or display, electronic securities transactions between specific parties. Given the hard-to-read nature of FIX in its various forms, as well as the lack of an agreed-upon representation standard for such transactions, this allows domain experts to quickly grasp the nature of a specific transaction. Conversely, it may be used to "prototype" or design transactions before they are performed. Currently, this is accomplished by creating several models using the metamodel in a sequential manner. With a simple extension, such as adding a *Time* field to all interaction entities, it would also be possible to model subsequent interactions in a single model, as the metamodel allows a (virtually) unlimited amount of entities to be placed in a single model. The solution domain best suited to this design would be the FIX protocol itself, possibly with implementation-specific extensions, or the domain-specific language given in the Appendix. An example of this use, modeling the interaction example in [14], is given in Section 5.2.

Chapter 5 Examples

In this chapter, we will demonstrate the expressiveness of our DSL by modeling several examples of FIX-based interaction from other sources. Using the Eclipse Modeling Framework, we have created a model editor from our metamodel. Using this editor, which functions as a plug-in within the Eclipse environment, one can create models of electronic trading interactions by intuitively placing Parties on a canvas and drawing Interactions between them.

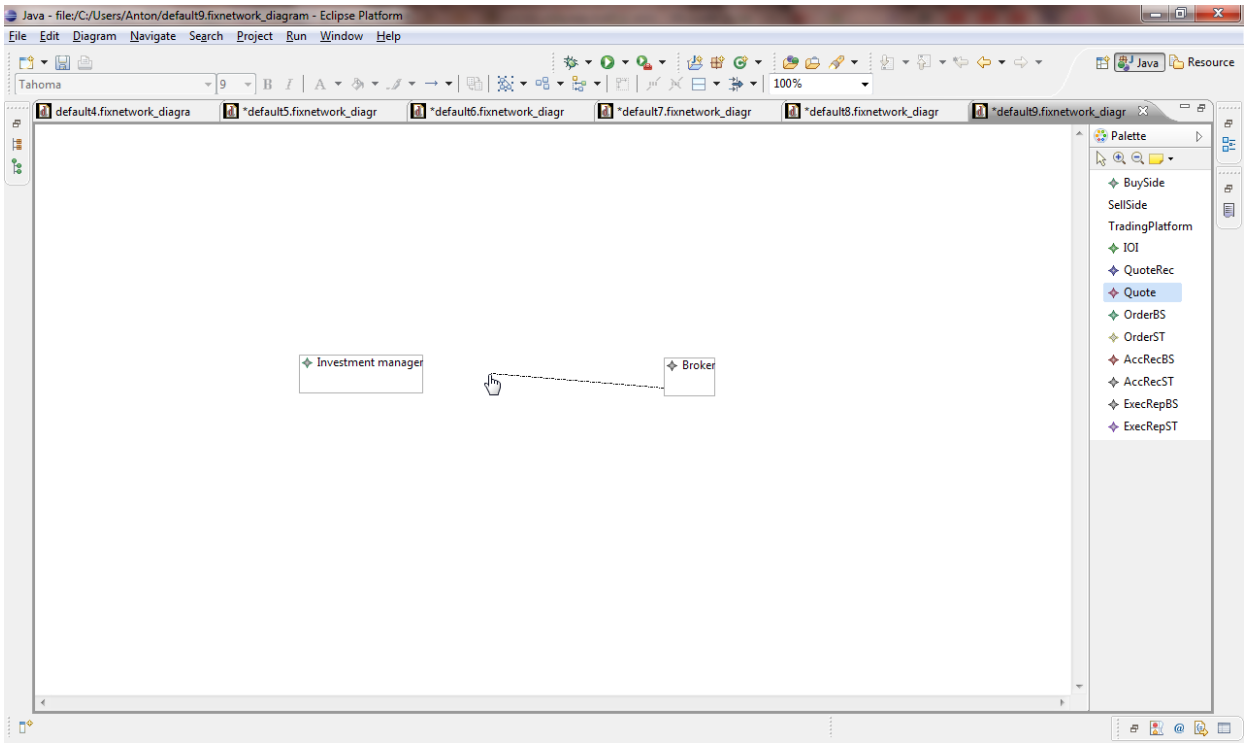


Figure 3: “FIXNetwork” editor

The editor enforces the constraints defined in the metamodel: interactions are only valid between specific entities, and directionality is enforced – Quote arcs, for instance, originate exclusively at Sell-Side entities, as per the semantics described in Section 3.

It is important to note the distinction, as noted in Sections 4.2.1 and 4.2.2, between displays of *generalized interaction* and those of *actual interaction*; the former are intended to display the range of connection and interaction possibilities allowed by a specific implementation (as one would find in its documentation), the latter is a means of demonstrating the various steps of a specific transaction as it took place. In other words, *generalized interaction* provides a template for use of a specific implementation, whereas *actual interaction* is a “logbook” of FIX message exchange. One example of each will be provided.

5.1 Generalized Interaction

An example of generalized interaction is given by [1] and reproduced in chapter 3 (Figure 1). It shows the basic order process as well as the parties involved. As a pre-trade interaction,

the distribution of market data is outside the scope of our domain, and as the role of the FIX in market data propagation is inferior to that in trade flow[low latency report, FIX website, Bloomberg website], the “market data” arcs and “financial content providers” entity have been removed. Furthermore, the “Verification” step is of a post-trade nature (and also accomplished through settlement-related protocols such as SWIFT), and is therefore removed as well. The relevant interaction represented in Figure 1 can therefore be modeled as follows:

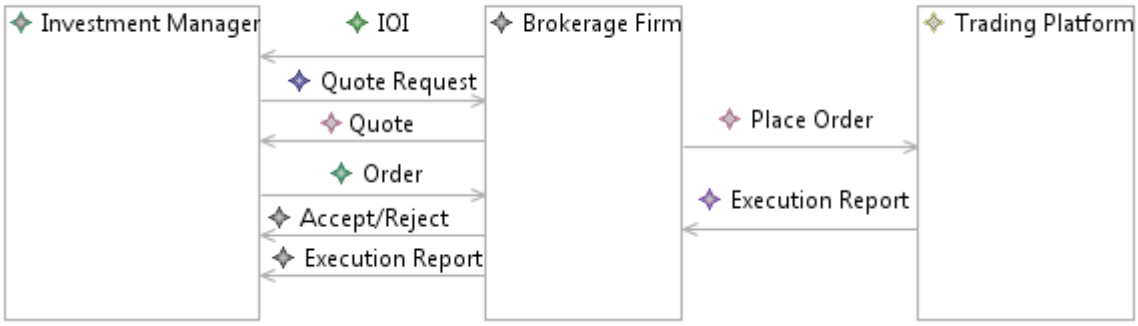


Figure 4: modeling generalized interaction

5.2 Actual Interaction

The FIX organization provides an example of FIX message exchange in [14], where a broker facilitates the sale of securities between two of his clients’ portfolios (notably not involving a trading platform). The transaction comprises four steps: first, Customer A sends an Order message to his broker, notifying his desire to sell 1000 shares of ‘Cheung Kong’, and receives an Execution Report message in return, to notify its order has been accepted. Subsequently, the Broker sends IOI messages to a multitude of clients, through a “FIX Network” – an implementation-dependent means of addressing multiple parties, as FIX usually functions only bilaterally, as described in Section 2. In our model, this network is represented by a single Buy-Side Entity. In step 3, out of this network, a single party (“Customer B”) responds to the IOI with an Order to buy the securities offered (at a price determined by the broker, in the IOI message). The broker accepts the sale trough an Execution Report message. In step 4, finally, the broker sends Execution Report messages to both parties, indicating the order as filled.

Figure 5: the customer indicates its willingness to sell from its portfolio (price not given)

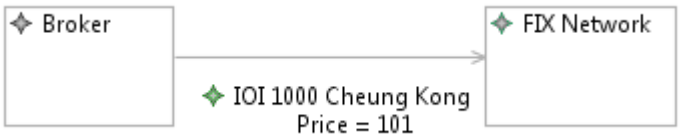


Figure 6: the broker informs his clients (through an unspecified FIX Network)

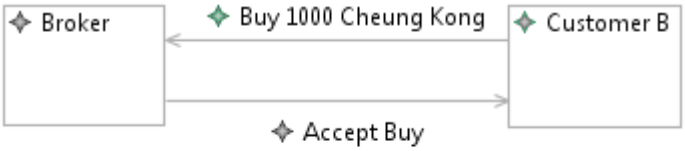


Figure 7: one customer decides to buy, and notifies the broker

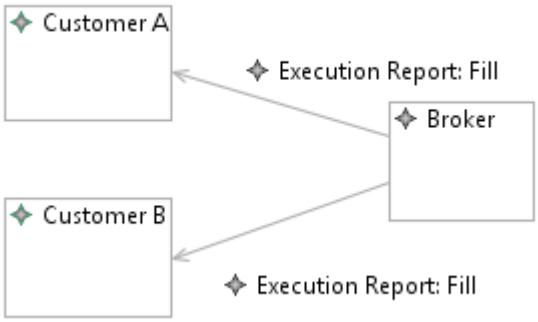


Figure 8: the broker notifies both parties of the transaction

Chapter 6: Conclusions and Future Work

It has been demonstrated that the electronic securities trading environment lends itself fairly well to domain-specific modeling, and that the FIX Protocol forms a suitable basis for a domain-specific language by incorporating in its design the numerous interactions typically expected in the trading of electronic securities. The protocol, while including numerous low-level mechanisms and concepts (somewhat alleviated in version 5.0) and being subject to a diverse array of specific implementations, can be used to define higher-level *interactions* as an abstraction of its workings. In this way, it is possible to capture domain-specific knowledge and implement a visual model to quickly demonstrate FIX-enabled connection arrangements as well as visualize message exchange between trading parties.

This analysis has been fairly limited in scope, as it aimed to capture the essence of electronic trading in a simple model. With regards to scope, we have based our analysis mainly on version 4.0 of the FIX protocol, thus predating the drastic expansion in version 5.0, as well as the more gradual changes in version 4.1 through 4.3. Further work may also be done to incorporate specific interactions pertaining to foreign exchange, derivatives and other more exotic financial instruments in which FIX has gained currency since its inception as an equity-specific protocol. Plus, pre-trade (such as market data) and post-trade (clearing and settlement) interaction may also be explored in future research.

Finally, in order to arrive at a full-fledged domain-specific language, it would be necessary to address the shortcomings mentioned in Section 4.2: provide evaluation semantics and a domain-specific processor to generate instances of a possible solution domain. This would entail considerably more effort than expended on this analysis, but should be possible based on the domain concepts from Section 3 and the metamodel from Section 4. In doing so, it might be necessary to expand the scope of the original analysis in a fashion described in the previous paragraph. This depends on the intended solution domain.

Appendix: a Textual DSL for FIX Messages

Concurrent to this analysis, a Domain-Specific Language, modeled after COBOL, was developed based on the FIX messaging standard. It may be used to render the exchange of FIX messages more readable to humans (compared with FIXML or the traditional *<tag=value>* form) or to quickly model specific message exchange between parties. It differs from our visual DSL in that it does not allow the modeling of generalized interaction arrangements.

1. The Session

As already mentioned in the previous section, the session provides the context of the FIX transmission, and can therefore be seen as a "code block" surrounding "statements" – the FIX messages. No statements are allowed to exist outside such a block, just as no FIX messages may be sent when there is no session. The session definition also serves to identify the parties involved as *Initiator* and *Acceptor*. There is always one Initiator, but there can be many Acceptors. As the mechanism of one-to-many communication is not defined in the FIX specification itself, and either occurs through the establishment of many one-to-one sessions or through a FIX Hub or Exchange, we will not explicitly define how such a one-to-many session is established.

Sessions are defined by the **SESSION…END SESSION** structure. The exact syntax is as follows:

SESSION FROM [INITIATOR] TO [ACCEPTOR] {statements} END SESSION

Where {statements} is optional and all other fields mandatory.

2. Initiator and Acceptor

Though the identities of initiator and acceptor have been defined in the *SESSION* statement, and most FIX messages have a predefined origin of either buy-side or sell-side firm, there is no predefined relationship between these nomenclatures. In other words, even though it is intuitively the buyer that initiates a session with the seller, the reverse may also be true [14]. Hence, it is necessary to explicitly state the origin of each message in the session, given the full-duplex nature of communications.

To this end, messages sent by the Initiator are prefixed by **I:** and messages sent by the Acceptor are prefixed by **A:**.

3. Grouping of similar tag fields

Given the large number of (optional) tag fields in each message definition, we have decided to group redundant or complementary fields into an aggregate abstracting the different fields for the purpose of syntax definition. For instance, a security can be identified by the fields *Symbol*, *SecurityID*, *SymbolSfx*, *IDSource*, *Issuer* and *SecurityDesc*, with only the *Symbol* field being mandatory. We believe that, without loss of generality, we can use *Symbol* in lieu of these fields. In the FIX 5.0 documentation [36] this grouping is explicitly stated: for instance, the "Instrument component block" to refer to the group of fields identifying the security in question.

Similarly, in quote messages, bid and order prices and quantities are grouped as well. Also, *TimeInForce* and *ExpireTime* are grouped.

4. Message IDs

As briefly touched upon in the previous section, the FIX protocol makes extensive use of IDs to allow messages to refer to other messages for status or modification purposes. Generation of these IDs is left to the implementation, as long as unicity is maintained for at least a 24-hour period [2]. Due to their essential nature we maintain these IDs in our DSL, though simplifying them somewhat. For one, the FIX protocol maintains a distinction between client-supplied ID and broker-supplied ID, providing both the *ClOrdID* and *OrderID* fields. While these are arguably important in a production situation (driven by the need to maintain unicity at the broker's, who likely handles several clients concurrently), we believe the distinction is not material to our analysis, and can hence be abstracted as a single ID.

Furthermore, to improve readability and reduce ambiguity, we use a statement's line number as the ID of the message represented. A line number can thus serve as *AdvRefID, ClOrdID, ExecID, ExecRefID, IOIid, IOIRefID, OrderID, OrigClOrdID, AllocID, RefAllocID, QuoteID, QuoteReqID* and *ListID.*

5. Replacing or cancelling messages

The FIX protocol provides for the possibility of cancelling or altering ("replacing") previously sent IOIs, Advertisements and Execution Report messages by resending a message of the same type with a the TransType (IOITransType, AdvTransType or ExecTransType) set to CANCEL and REPLACE/CORRECT, respectively, as well as any fields that need to be altered. In the interest of readability, we have chosen to represent these messages not by their "normal" statement keyword, but with the REPLACE and CANCEL statements. To "replace" an IOI, for instance, a new IOI message is sent with *IOITransType = REPLACE* and *IOIRefID* set to the original IOI's ID; in our DSL, this would be represented by "*REPLACE INDICATE <linenum> WITH…"*

Please note that this mechanism does not apply to orders. Orders have a different cancelling/alteration mechanism, using the *Order Cancel…* messages.

6. Messages

Messages are modeled by single-line statements. Given the diverse use of mandatory and optional fields for each message, every message type will be defined individually. We have chosen to use a COBOL-like syntax for statements - using verbosity to represent statements

as English-language sentences – instead of a more C-like syntax representing statements as functions. Furthermore, statements in our DSL do not have a return value. Analogous to the FIX protocol, errors and confirmations are signaled through messages returned by the sender. Statements are non-blocking by default, with synchronization provided by the **WAIT FOR** statement.

Please note that all FIX messages have standard headers and trailers affixed. These contain information mostly pertaining to the session protocol (abstracted through the SESSION statement), for instance to identify duplicate messages, and a simple addressing system. We consider these fields irrelevant to our analysis, therefore, the contents of these headers and trailers will not be included in our DSL syntax.

3.1 Indications of Interest

IOI's are represented by the "INDICATE TO" keyword, followed by BUY or SELL, according to the following format:

INDICATE TO[BUY|SELL] [<quantity> OF] [Symbol] {FOR <price>} {IN <currency>} {UNTIL <timestamp>} {AGENCY|PRINCIPAL}

3.2 Quote Requests

Quote Requests are represented by the "REQUEST QUOTE FOR" keyword including the symbol and optionally the quantity and side (buy/sell), as follows:

REQUEST QUOTE FOR {BUY|SELL} {<quantity> OF} [Symbol]

3.3 Quotes

Quotes are represented by the "QUOTE" keyword and mandatorily the symbol and bid price, with several optional keywords, as follows:

QUOTE {<quantity> OF} [Symbol] [FOR <price>] {UNTIL <timestamp>}

3.4 New Order – Single

Single orders are represented with the "ORDER TO" keyword, followed by the Side, quantity, Symbol, and, when applicable, references to the quote/IOI the deal is responding to.

ORDER TO [BUY|SELL][<quantity> OF] [Symbol]{QUOTED IN <linenum> | INDICATED IN <linenum>} {IN <currency>}

3.5 Execution Reports

Execution reports are semantically different from most other statements, as they don't imply any action to be taken and just report on a previous order. Nevertheless, they are treated like any other statement in our DSL and represented by the "REPORT ON" keyword, followed by the order's ID (i.e. its line number) and its status (New, Partially Filled, Filled, Done for Day, Canceled, Replaced, Pending Cancel/Replace, Stopped, Rejected, Suspended, Pending New, Expired or Calculated), Side, amount of shares (*LastShares* field) and last price.

REPORT ON [linenum] STATUS IS [New|Filled|…] [BOUGHT | SOLD] [<quantity> SHARES] [FOR <price> <settlement currency>]

3.6 Don't Know Trade

Just like the *Execution Report*, the Don't Know Trade message merely indicates a condition, not asking for any action to be taken. Most of the fields are just copied over from the original order message, and hence not used.

DON'T KNOW TRADE {linenum} BECAUSE [<reason>]

3.7 Order Cancel/Replace Request

[what to do with all the fields? There's too many][Side and symbol included, but must stay the same as previous order. Price and quantity can be changed, so we keep these fields to create a full sentence][beware of ambiguity with "our" REPLACE keyword, see above]

REPLACE ORDER [linenum]TO [BUY|SELL]{quantity}[Symbol] {FOR <price>} {UNTIL <expiretime>} {INSTRUCT ExecInst}

3.8 Order Cancel Requests

Order Cancel Requests are represented with the "*CANCEL ORDER*" keyword, followed by the order ID and parameters indicating whether the order should be fully cancelled or its quantity reduced.

CANCEL ORDER [linenum][FULLY | {TRUNCATE TO <quantity}]

3.9 Order Cancel Reject

Just like *Execution Report* and *Don't Know Trade*, this message serves to indicate the outcome of a process – in this case, a request to cancel an order. It is represented by the "*REJECTED CANCEL*" keyword:

REJECTED CANCEL [linenum] {BECAUSE <reason>}

3.10 Order Status Request

This message is represented using the "*REQUEST STATUS ON*" keyword. While the FIX message mandates the *Side* and *Symbol* fields, they must be the same as those in the original order, and we refer to that using the *ClOrdID* field. They are hence considered redundant and not included in our syntax.

REQUEST STATUS ON [linenum]

References

1 Oxera. *What are the benefits of the FIX protocol? Standardising messaging protocols in the capital markets.* December 2009. Available at http://www.fixprotocol.org/news/publications.shtml

2 The FIX Committee. *Financial Information Exchange Protocol (FIX) Version 4.0*. January 1997. Available at http://www.fixprotocol.org/specifications/FIX.4.0

3 Euronext. *EURONEXT FIX Trading Interface, Version 2.0.1.* October 2006. Available at http://www.integration.euronext.com/fic/000/020/502/205024.pdf

4 A. Benn. *The Unseen Wall Street of 1969-1975*. Quorum Books, 2000

5 W. Wells. Certificates and Computers: The Remaking of Wall Street, 1967 to 1971. *The Business History Review*, Vol. 74, No. 2 (Summer, 2000) pp. 193-235

6 D. Donald. *The Rise and Effects of the Indirect Holding System: How Corporate America Ceded its Shareholders to Intermediaries*. Doctoral thesis adaptation, University of Frankfurt, 2007

7 D. Chan, F. Fontan, S. Rosati and D. Russo. *The Securities Custody Industry*. European Central Bank Occasional Paper Series, August 2007

8 B. Mizrach and C. Neely. *The Transition to Electronic Communications Networks in the Secondary Treasury Market*, Federal Reserve Bank of St. Louis Review, November/December 2006, 88(6), pp. 527-41

9 F. Black. Toward a Fully Automated Stock Exchange. *Financial Analysis Journal*, November-December 1971

10 N. Economides and R. Schwartz. Electronic Call Market Trading. *Journal of Portfolio Management,* vol. 21, no. 3, pp. 10-18 (1995)

11 I. Domowitz. A taxonomy of automated trade execution systems. *Journal of International Money and Finance*, issue 12, pp. 607-631 (1993)

12 Bank for International Settlements. *The implications of electronic trading in financial markets*, 2001

13 M. Massimb and B. Phelps. Electronic Trading, Market Structure and Liquidity. *Financial Analysts Journal*, January-February 1994

14 FIX Protocol Limited. *Financial Information eXchange General Conference Hong Kong, March 30, 2000*. Available at http://www.fixprotocol.org/specifications/TechDoc-Beginner

15 FIX Protocol Limited website. http://www.fixprotocol.org

16 FIX Protocol Limited. *FIX Protocol Open Forum – London*, 2000. Available at http://www.fixprotocol.org/specifications/TechDoc-Beginner

17 R. Pierce. *Transitioning to Advanced Versions of Messaging Standards*. Townsend Analytics Ltd. / Archipelago LLC, 2001.

18 A. van Deursen, P. Klint and J. Visser. *Domain-Specific Languages: An Annotated Bibliography.* CWI, Amsterdam, 2000

19 M. Mernik, J. Heering and A. Sloane. When and How to Develop Domain-Specific Languages. *ACM Computing Surveys, Vol. 37, No. 4, December 2005, pp. 316-344*

20 J. Gray and G. Karsai. An Examination of DSLs for Concisely Representing Model Traversals and Transformations. *Proceedings of the 36th Hawaii International Conference on System Sciences*, 2003

21 F. Latry, J. Mercadal and C. Consel. Processing Domain-Specific Modeling Languages: A Case Study in Telephony Services. In: *Generative Programming and Component Engineering for QoS Provisioning in Distributed Systems*, 2006

22 J. Luoma, S. Kelly, J. Tolvanen. Defining Domain-Specific Modeling Languages: Collected Experiences. In: *Proceedings of the 4th OOPSLA Workshop on Domain-Specific Modeling(DSM '04), Vancouver, Canada*, 2004

23 M. Karlsch. *A model-driven framework for domain specific languages demonstrated on a test automation language*. Master thesis, Hasso-Plattner-Institute of Software Systems Engineering, Potsdam, Germany, 2007

24 S. Kelly. *Domain-Specific Modeling – 76 cases of MDD that works*. MetaCase, 2009

25 S. Kelly and R. Pohjonen. *Worst Practices for Domain-Specific Modeling*. IEEE, 2009

26 B.R.T. Arnold, A. van Deursen, M. Res. An algebraic specification of a language for describing financial products. In: *ICSE-17 Workshop on Formal Methods Application in Software Engineering*, 1995

27 S.P. Jones, J. Eber and J. Seward. Composing contracts: an adventure in financial engineering. In: *ICFP '00 Proceedings of the fifth ACM SIGPLAN international conference on functional programming*, 2000

28 A. van Deursen, P. Klint. Little Languages: Little Maintenance? *Journal of Software Maintenance: Research and Practice,* 10, 75-92, 1998

29 The FIX Committee. *FIX Protocol 4.0 Release Notes*, 1996. Available at http://www.fixprotocol.org/specifications/FIX.4.0

30 FIX Protocol Limited. *Financial Information Exchange Protocol (FIX) Version 5.0 Volume 1*, 2006. Available at http://www.fixprotocol.org/specifications/FIX.5.0

31 Securities and Exchange Commission. *Rule 300a, 17 CFR 242.300(a).* Securities and Exchange Commission, 1998

32 LAVA Technology. *LavaFlow ECN FIX Specifications,* 2010. Available at https://www.lavatrading.com/solutions/LavaFlow_FIX_Specs_1-7.pdf

33 Securities and Exchange Commission. *Alternative Trading System ("ATS") List*, 2011. Available at http://www.sec.gov/foia/docs/atslist.htm

34 European Parliament and Council of the European Union. *Directive 2004/39/EC*, article 4, 2004. OJ L 145, 30.04.2004, p. 1

35 A. Bowley, K. Nandwani. *Nomura NX FIX Specifications*. Nomura, 2010

36 FIX Protocol Limited. *Financial Information Exchange Protocol (FIX) Version 5.0 Volume 3*, 2006. Available at http://www.fixprotocol.org/specifications/FIX.5.0

37 FIX Protocol Limited. *Financial Information Exchange Protocol (FIX) Version 5.0 Volume 4*, 2006. Available at http://www.fixprotocol.org/specifications/FIX.5.0