LOBSTER: Limit Order Book Reconstruction System*

Ruihong Huang†
Humboldt-Universität zu Berlin

Tomas Polak ‡
Humboldt-Universität zu Berlin.

December 27, 2011

Abstract

The rise of order-driven markets in recent years created considerable challenges for researchers, who have to cope with extremely large amounts of data produced daily by the markets. It is our goal to spare academic researchers the tedious task of technical preprocessing of the data and thus enable them to focus on economic research. Our order book reconstruction system LOBSTER is based on the generalized order-processing algorithm common for majority of order-driven markets and so it is easily adjustable to process data produced in any order-driven market. Efficient data structures result in very convincing performance, with large datasets produced on the fly. Currently LOBSTER uses ITCH data from NASDAQ to accurately replicate the limit order book for any NASDAQ-traded stock to any desired level. Data from more venues will be added in the future. The system is accessible via the Internet, which makes it very convenient for researchers from around the world.

Keywords: Limit order market, Message data, High-frequency data

JEL classification: C88

1 Introduction

An electronic limit order market is an order-driven market which automatically collects orders from traders in a centralized limit order book (LOB) and matches corresponding buy and sell orders based on specific priority rules, very often the *price-time priority rule*. Currently, most

*For helpful comments and discussions we thank the entire LOBSTER development team: Jonas Haase, Gustav Haitz, Nikolaus Hautsch and Gagandeep Singh. Created with strong support of Research Data Center at CRC649: Economic Risk.

†Institute for Statistics and Econometrics, Humboldt-Universität zu Berlin. Email: ruihong.huang@wiwi.hu-berlin.de Address: Spandauer Str. 1, 10178 Berlin, Germany.

‡Research Data Center at CRC649: Economic risk and Institute for Statistics and Econometrics, Humboldt-Universität zu Berlin. Email: polaktox@wiwi.hu-berlin.de.

equity exchanges around the world are either pure electronic limit order markets, e.g. NYSE Arca, BATS, Euronext, Australian Stock Exchange (ASX) and Direct Edge, or at least allow for customer limit orders in addition to on-exchange market making, e.g. NASDAQ, NYSE and the London Stock Exchange (LSE). The traditional monopolistic power of market makers in the area of liquidity provision through quoting on both sides of the market has been strongly restricted, if not completely eliminated. Instead, the important task of providing liquidity is now assigned to the complex trading interactions enabled by the emergence and disclosure of the LOB. Hence, the state of the LOB is extremely important for practitioners, because it allows them to optimize their trading strategies, but also for researchers who analyze trading activity in these markets and try to interpret the underlying economic motivation.

One of the most prominent market structure developments in recent years is high frequency ("HF") trading. HF traders in general employ extremely quick and sophisticated computer programs for generating, routing and executing orders. They establish and liquidate positions in very short time-frames by submitting numerous orders and cancelling non-executed orders shortly after submission. As a consequence, the trading volume grows, orders shrink in size and the pace of LOB updating is beyond human perception, requiring nanosecond precisions. The volume of information about orders recorded by market organizers therefore dramatically increases.

More than ever before researchers today face the challenge of working with real datasets on micro-structure level of financial markets. They are in general not difficult to obtain, but very difficult to process. One way of going about it is acquiring snapshots of historical LOB data. But this is very impractical, because such datasets are usually very large and contain only incomplete information[1]. The second option is to acquire much better compressed raw message stream data, like TotalView-ITCH and Multicast PITCH data. Because message data record all visible order activities, i.e. limit order submissions, cancellations and executions, they can be used to reconstruct the historical LOB up to any required precision (level). However, this creates a different type of challenge; it is necessary to reconstruct the LOB using the same rules that were used by the matching algorithm applied by the exchange. Although simple in principle, such algorithms need to take into account market-specific issues and, considering the large volume of data, work extremely efficiently.

In this paper, we present you the LOBSTER, a program framework for reconstructing LOBs as well as extracting order flow information from historical message stream data. Note that the fundamental limit order activities are quite similar across different limit order markets, though the specific trading rules can be very different. We modularize LOBSTER to processes limit order activities translated from messages instead of messages themselves, so that it can be easily adapted to data from new limit order markets with just a few modifications. The underlying data structure in these modules is highly optimized and their programs are exhaustively tested

[1]Snapshots are made at regular time intervals, e.g. every second, which means that multiple changes within this time intervals may be omitted. Should snapshots capture *all* changes in the LOB to *all* levels the size of files to store these datasets would be too large for practical purposes.

to guarantee the reliability of the output data and the efficiency of the entire system.

Currently we implement a translation module for NASDAQ TotalView-ITCH message and a web-based interface. Researchers around the world can easily access our program through `http://lobster.wiwi.hu-berlin.de` and download NASDAQ LOB data reconstructed on the fly. Moreover, a related forum facilitates general discussion on empirical analysis of LOB, as well as bundles of small programs and useful tools in both R and Matlab.

The remainder of this paper is structured as follows. In Section 2, we take the NASDAQ TotalView-ITCH as an example to illustrate the structure of order-related messages in message stream data. Section 3 gives the overview of the design of the LOBSTER. We discuss in detail on two implemented modules, the LOB Constructor and the Order Tracer, in Section 4 and 5, respectively. Finally, Section 6 concludes.

2 Message Stream Data

As we already discussed above, unlike historical snapshots of the LOB, message-type data allow reconstructing the full LOB and observing its dynamics with maximum precision. This is indeed extremely valuable for academic research, giving the possibility to study all aspects of trading in full detail. In this section, we will take the NASDAQ TotalView-ITCH 4.0 as an example to illustrate the structure of messages.

The original NASDAQ TotalView-ITCH is a direct data feed that contains market messages for all *submissions*, *cancellations* and *executions* of limit orders, as well as *executions of hidden orders*. TotalView-ITCH 4.0 data is the binary version of the TotalView-ITCH data introduced in November 2008, following the ITCH 3.0 format. This format is ready for nanosecond time stamps[2] and contains longer order and trade identification numbers allowing to mark up to 18 quadrillion messages per day.

Considering the enormous number of messages generated by the activities in markets every day, the TotalView message stream is designed to present information in a parsimonious way, reducing redundancy in the records. Table 1 illustrates the message types related to the instructions carried by orders. Whenever a market participant submits a new order, its details, such as order ID, limit price and size, are recorded in a *submission* message. If the market participant chooses to reveal her identity to other market participants the *submission* message contains also her Market Participant Identification (MPID). All subsequent changes to the submitted limit order are also recorded in the form of messages; messages reporting partial cancellations or executions (partial or total) contain the same order ID as the original *submission* messages and the cancelled or executed quantity. If the limit order is totally cancelled (deleted), TotalView records only the corresponding order ID. Finally, messages reporting executions of hidden orders contain only the price and the executed quantity, but *not* the total size of the originally submitted hidden orders.

[2]The time resolution of trading in NASDAQ is still in milliseconds. Therefore, the time stamps in the current data set have still millisecond precision rather than nanosecond.

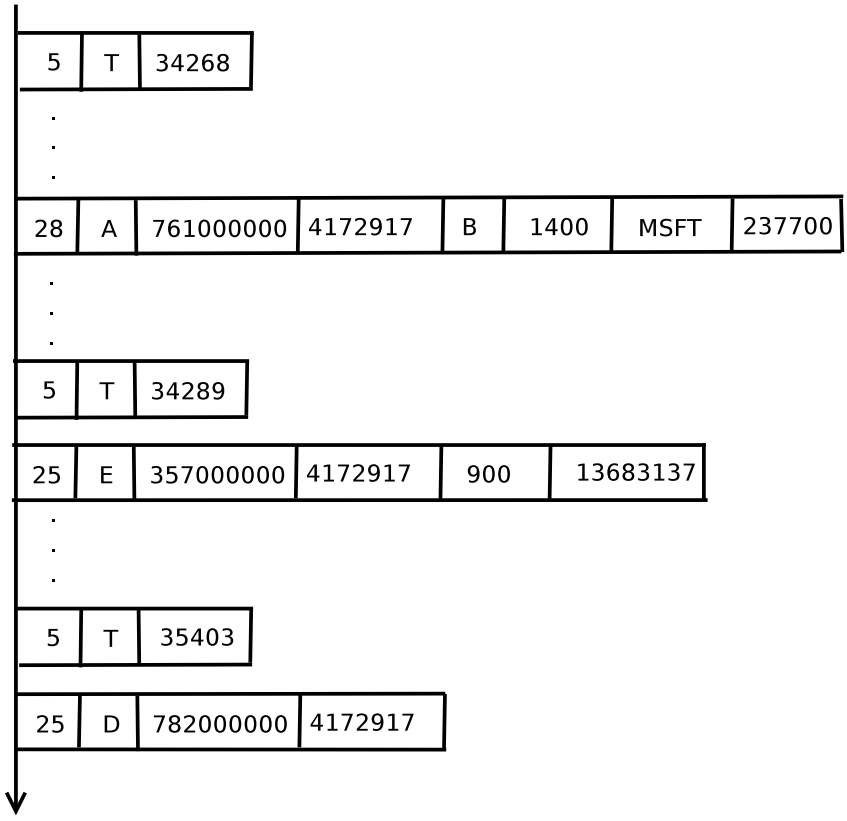


Figure 1: A sequence of messages related to the same limit order. The first part of each message contains information about the length of the message. The second part is the message type. Time stamp messages "T" record the number of seconds after midnight. The third part of messages (except for time stamp messages) contains the time in nanoseconds since the last time stamp message. The fourth part is the order ID. For type "A" messages, the remaining fields contain: trade direction (buy/sell) indicator, order size, stock ticker, limit price. Type "E" messages further contain information about the executed size and a unique matching number.

Table 1:
TotalView-ITCH: order-related messages.

TotalView-ITCH includes messages representing *submissions*, *executions* and *cancellations* of limit orders, as well as *executions of hidden orders*. A *submission* type limit order message may or may not contain information identifying the market participant, who submitted the order - Market Participant Identification (MPID). A type "P" message reports the execution price and immediately traded quantity, i.e. only that part of the hidden order, which is currently being executed against an incoming order. Thus no information about the remaining unexecuted part of the hidden order is revealed.

Instruction	Type	Limit/Hidden Order				Canc./Exe.
		ID	Price	Size	MPID	size
Limit order submission	A	✓	✓	✓		
Limit order submission with MPID	F	✓	✓	✓	✓	
Limit order execution	E	✓				✓
Limit order cancellation (partially)	X	✓				✓
Limit order cancellation (totally)	D	✓				
Hidden order execution	P	✓	✓			✓

However, this parsimony raises challenges for researchers who intend to use TotalView message data. Most messages contain only incomplete information for the corresponding order. Figure 1 illustrates a sequence of messages related to the same limit order. Only the "A" message, which carries information about limit order submission, contains information about limit price, size and trade direction of the limit order. Therefore, before type "E" or "D" messages can be used by the algorithm to update the LOB, we need to "trace back" the corresponding "A" message to retrieve the information. Note also that ITCH records time stamps in two parts. The first part of a time stamp is carried by a type "T" message, which records the number of seconds after midnight. The second part is the number of nanoseconds since the last recorded second, recorded in the third position of each message.

Compared to the Trades and Quotes Database (TAQ) released by NYSE, which contains the best quotes and the corresponding depths, message data has richer information. It records order activities, which pooled together comprise the quote prices and depths. Thanks to this superiority of information content message data can be used to reconstruct the LOB up to any quote level. But even when looking only at the best quote and depth, message data is richer than the usual data obtained from the TAQ database. Message data contain information about limit orders which are cancelled shortly after the submission. These limit orders are typically submitted in order to detect hidden liquidity inside the spread rather than to provide liquidity (see e.g., Hasbrouck and Saar, 2009). Therefore, it is not surprising that TAQ ignores them as shown in Figure 2. However, these orders are crucial for some studies, e.g the analysis of high frequency trading strategies and hidden order submission strategies.

The currently implemented LOBSTER is connected to a storage facility containing over 5 TB of historical TotalView-ITCH data, in ITCH 3.0 format from the period Jan 2007 to Apr 2009 and ITCH 4.0 format from the period May 2009 to Dec 2010. This dataset contains

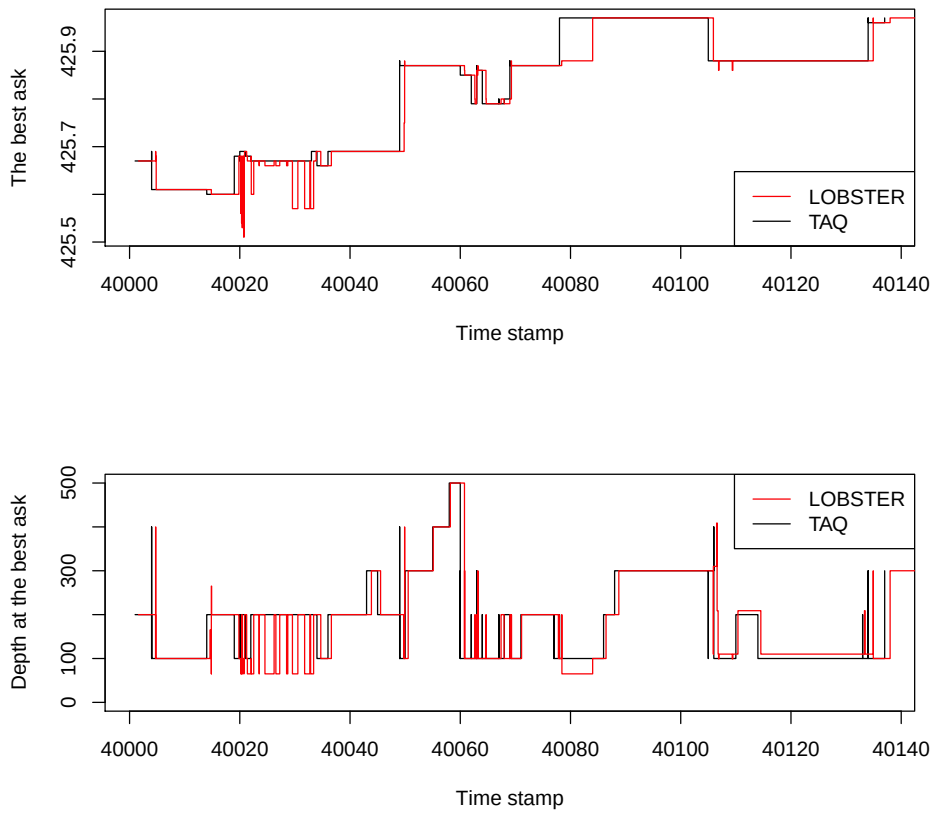


Figure 2: Comparison of TAQ and one-level LOB generated by LOBSTER. The time stamp is the number of milliseconds after midnight.

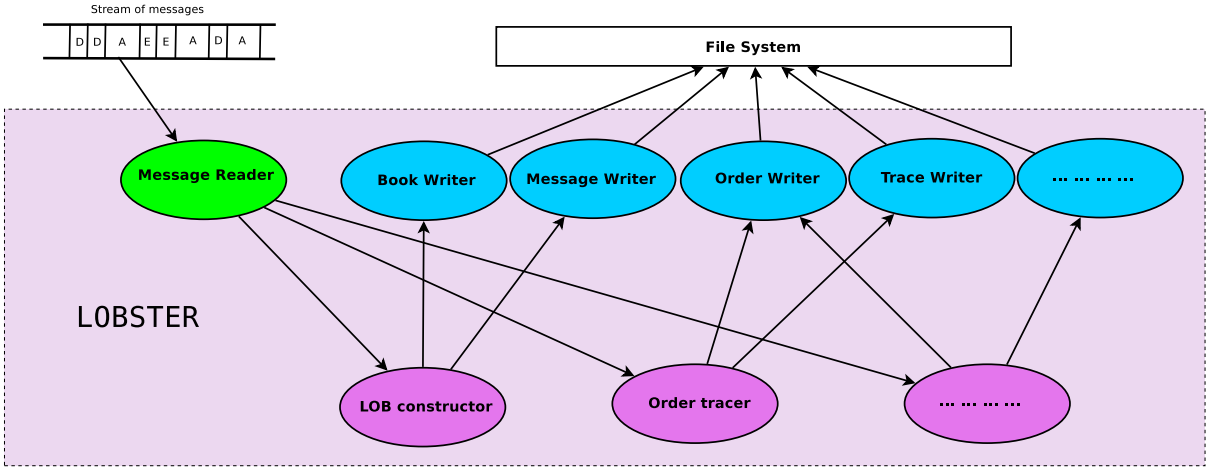


Figure 3: The overview of LOBSTER system. Three types of modules are used in general. Readers (green) read the source data into the system; data Processors (purple) retrieve the information; Writers (blue) write the output into the file system. For the sake of flexibility Readers and Writers are normally implemented as interfaces.

only limit order messages. Other messages such as imbalance data events and administrative messages have been cleaned out.

3 Overview of LOBSTER

The main goal of LOBSTER is to provide a reliable, efficient and flexible platform for researchers to retrieve information from parsimonious message data. Relying on the object-oriented concept, we designed LOBSTER as a modular system with three types of modules: *Readers*(green), *data Processors*(purple) and *Writers*(blue) as shown in Figure 3.

Reader translates data from an external source to a stream of order events, which is then processed by data Processors. Based on a unified abstract interface, different readers can be created for reading different input formats, e.g. from different stock exchanges. But also an order-flow simulator can act as a "Reader", provided that the generated data have the required format. Currently the system contains an ITCH message Reader for reading binary ITCH files from the storage facility and a test version PITCH Reader for BATS.

Data Processor is the core of the system responsible for extracting the information required by users from the order flow generated by Reader. Connected to Reader by an abstract interface, the data Processors treat historical order flow, simulated order flow or hybrid order flow identically. This creates great potential for very effective testing of trading strategies.[3]

[3]For instance, it is well-known that the back-testing of trading strategies on historical data has a very serious drawback - there is no market feedback to the tested strategy and so it is very hard to estimate the market impact of the strategy. In this context, testing with a simulator allows implementing realistic feedback mechanisms to simulate market impact and thus provide credible assessment of the tested trading strategy.

We implemented a `LOB Constructor` that matches the limit orders coming from the Reader and updates the LOB. The accuracy and efficiency of this algorithm determine to a large extent the overall performance of the system. For this reason it was thoroughly tested and optimized. We shall discuss the currently implemented `LOB Constructor` in Section 4.

Moreover, we have also developed `Order Tracer`, whose beta version is currently being tested. `Order Tracer` traces relevant events for individual limit orders, such as the time of submission, partial/full execution and cancellation. It has been used for computing the lifetime of limit orders for some research projects (e.g. Hautsch and Huang, 2011a,b). We expect that in the near future more and more utilities for special research purposes will be added to the current framework.

Writer receives the reconstructed data and saves them to the file system. Currently the system contains four Writers; the `Message Writer`, which saves the event type and the corresponding order information into the file system, the `Book Writer`, which receives the current state of the reconstructed LOB for every order event and saves it, the `Order Writer`, which saves the characteristics (order ID, limit price, size, etc.) of limit orders, and finally the `Trace Writer`, which saves event-specific information, such as the event time, submission time and the type of order event, as generated by the `Order Tracer`.

4 Limit Order Book Reconstruction

4.1 Overview of the Reconstruction Procedure

Figure 4 summarizes the procedure of the LOB reconstruction. For arriving messages identified as limit order submissions, the system records in the order pool all relevant information including order ID, limit price, quantity, trade direction and MPID, if available. Once a cancellation or execution message arrives, the system first finds in the pool the corresponding previously recorded limit order submission by comparing the order IDs. After matching the two orders - incoming order and order stored in the order pool - the system records the remaining non-executed size of the limit order or deletes the order from the order pool altogether if the remaining size is zero. Finally, the system updates the LOB: the side in the LOB (bid or ask) to be updated is determined by trade direction (buy or sell) of the corresponding order; the level in the LOB is identified by the limit price; the new depth at this level is calculated by deducting the size of the effective quantity.

The remaining issue is the construction of the initial-state of the LOB before the aforementioned procedure can be applied. Note that in the TotalView-ITCH message data, the order ID of any limit order cancellation and execution message can always be found in a limit order submission message, which was recorded at an earlier time on the same trading day. This implies that all limit orders valid overnight, such as some good-to-kill orders, have been resubmitted by the system in the early morning. The NASDAQ trading system is in general open for new order

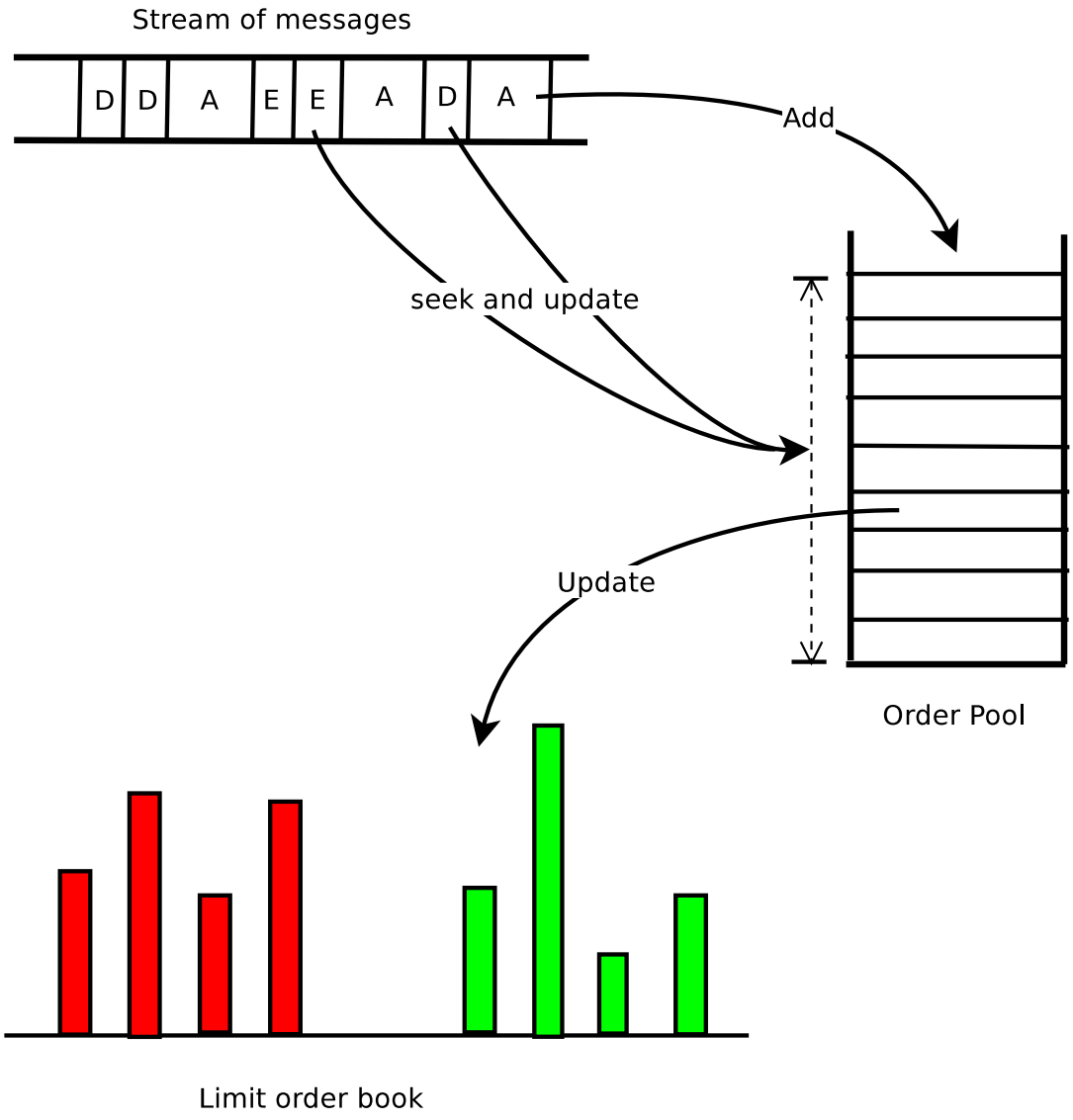


Figure 4: LOB reconstruction procedure. The algorithm employs an order pool to collect the limit order information. When an "A" (or "F") message comes in, it creates a limit order item in the order pool. When subsequently a message comes in indicating limit order cancellation ("X" and "D") or a limit order execution ("E"), the information about the price and size of the original limit order is retrieved from the order pool using common order ID.

instructions at 7:00 EST, even though continuous trading does not start until 9:30 EST. Because the TotalView-ITCH data set contains all messages, including messages submitted during the pre-trading period, our algorithm initializes the reconstruction with an empty LOB at the beginning of every day.

4.2 Implementation of LOB Constructor

As we discuss in Section 2, all TotalView-ITCH messages, except for messages containing limit order submissions, contain only partial information about the underlying limit orders. In order to update the order book, we need to "complete" the limit order information. Figure 5 shows in details how this is done in case of messages containing limit order executions. After reading an execution message, the system searches for the corresponding limit order inside the order pool using the order ID. If the search is successful, the information on the remaining size of the limit order is updated. The system then updates the LOB by changing the depth and quote on the corresponding level. Finally, the new order book and the corresponding message item are stored in the file system as output.

Figure 6 represents the final class diagram for our LOB Constructor. The system includes a unique `OrderBook` and `OrderPool` instance, which are controlled via a unique `BookConstructor` instance. After `MessageReader` objects reads a message from an external source, it creates a `Message` object and then uses it to update the limit orders inside the `OrderPool` object as well as quote and depth information in the `OrderBook` object according to the order event type. If the process is successful, it saves the output using a `MessageWriter` and an `OrderBookWriter` object. Note that the `Reader` and `Writer` are intent to be defined as abstract interfaces, rather than concrete classes, for increasing the flexibility of the format of output data.

4.3 Output of LOB Constructor

The `LOB Constructor` generates two output data files; one file contains the LOB data and the other one contains the corresponding order events. Table 2 shows a segment of reconstructed three-level LOB data for Google Inc. (with ticker GOOG in NASDAQ). It includes quotes and the corresponding depths up to the third best ask and bid, together with time stamps. The other file contains the corresponding order event. As shown in Table 3, it has five fields:

- `time`: time stamp; milliseconds after mid-night.
- `type`: event type; 1 for limit order submission, 2 for partial cancellation, 3 for total deletion, 4 for limit order execution, 5 for hidden order execution.
- `order ID`: a unique number assigned by the exchange for identification of orders.
- `size`: change of order size (in shares); for limit order submission it is the order size, for order execution it is the trading volume and for limit order cancellation it is the cancelled volume.

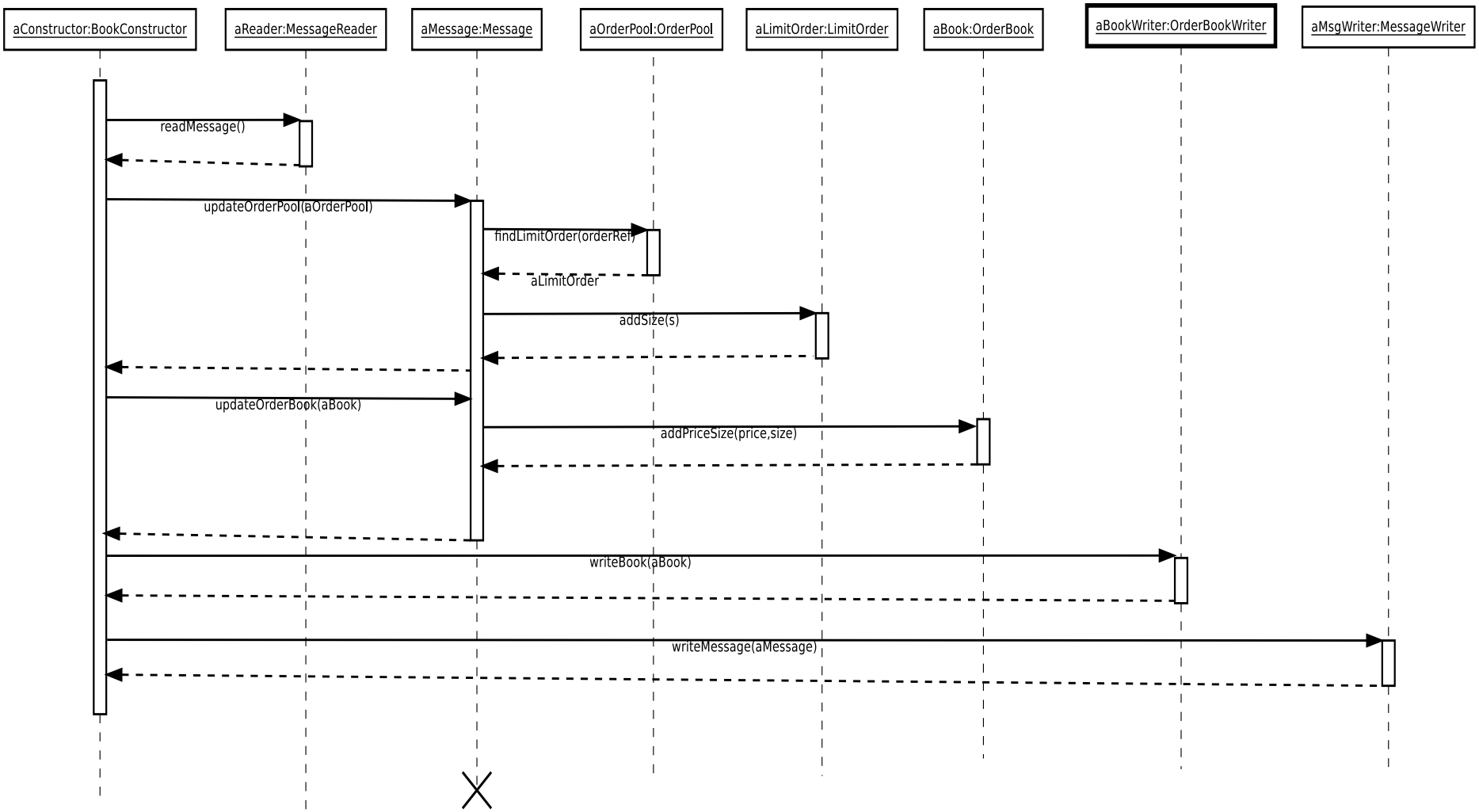


Figure 5: Sequential diagram for messages of limit order executions. The procedure begins with reading the message and creating a corresponding `Message` object. The object then communicates with the order pool and updates information about the underlying limit order. Simultaneously, it retrieves from the limit order any missing information, such as price. Using the complete information about the limit order, the `Message` object updates the order book state (the `OrderBook` object). Finally, the constructor writes the updated order book state and the corresponding message into the file system using the `OrderBookWriter` and `MessageWriter` objects.

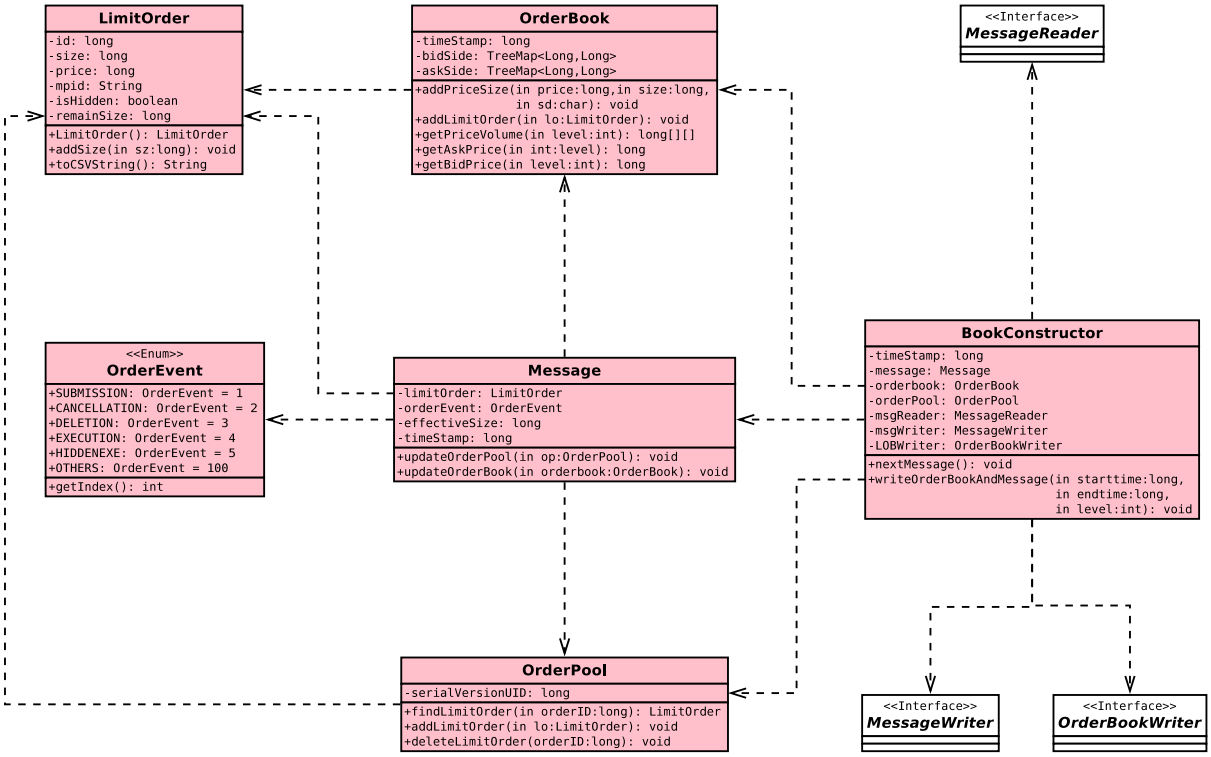


Figure 6: Class diagram of LOB construction. A `BookConstructor` object contains unique `OrderBook` and `OrderPool` objects, which are updated using incoming messages corresponding to `Message` objects. Moreover, there must be at least one `MessageReader` object for reading the input, an `OrderBookWriter` object and a `MessageWriter` object, which write the output into the file system.

Table 2:

LOB data generated by the LOB Constructor.

The sample contains a five second segment of three-level LOB data for ticker GOOG on July 1st 2009. The time variable is in milliseconds after the midnight. Price is in 0.01 of a cent and size in number of shares. The corresponding order events that update the state of the LOB are shown in Table 3.

Time	Ask price 1	Ask size 1	Bid price 1	Bid size 1	Ask price 2	Ask size 2	Bid price 2	Bid size 2	Ask price 3	Ask size 3	Bid price 3	Bid size 3
36000043	4231100	100	4227300	300	4231200	100	4223000	100	4231300	300	4222900	100
36000044	4231100	100	4227300	300	4231200	300	4223000	100	4231300	300	4222900	100
36000207	4229100	100	4227300	300	4231100	100	4223000	100	4231200	300	4222900	100
36000208	4229100	100	4227300	300	4231100	100	4223000	100	4231200	100	4222900	100
36000208	4229100	100	4227300	300	4231100	100	4223000	100	4231300	100	4222900	100
36003222	4229100	100	4227300	300	4231100	100	4223000	100	4231200	100	4222900	100
36003471	4229100	100	4227300	300	4231100	100	4222900	100	4231200	100	4221200	100
36004005	4229100	200	4227300	300	4231100	100	4222900	100	4231200	100	4221200	100
36004009	4229100	200	4227300	200	4231100	100	4222900	100	4231200	100	4221200	100
36004009	4229100	200	4222900	100	4231100	100	4221200	100	4231200	100	4219100	400
36004009	4229100	200	4222900	100	4231100	100	4221200	100	4231200	100	4219100	400
36004010	4229100	200	4222900	200	4231100	100	4221200	100	4231200	100	4219100	400
36004010	4229100	200	4227300	200	4231100	100	4222900	200	4231200	100	4221200	100
36004011	4229100	100	4227300	200	4231100	100	4222900	200	4231200	100	4221200	100
36004015	4229100	100	4227300	200	4231100	100	4223300	100	4231200	100	4222900	200
36004016	4229100	100	4227300	200	4231100	100	4222900	200	4231200	100	4221200	100
36004017	4229100	100	4222900	200	4231100	100	4221200	100	4231200	100	4219100	400
36004018	4229100	100	4222900	200	4231100	100	4222800	200	4231200	100	4221200	100
36004018	4229100	200	4222900	200	4231100	100	4222800	200	4231200	100	4221200	100
36004018	4229100	200	4227300	200	4231100	100	4222900	200	4231200	100	4222800	200
36004020	4229100	200	4227300	200	4231100	100	4222900	200	4231200	100	4221200	100
36004020	4229100	100	4227300	200	4231100	100	4222900	200	4231200	100	4221200	100
36004021	4229100	100	4227300	200	4231100	100	4223300	100	4231200	100	4222900	200
36004025	4229100	100	4223300	100	4231100	100	4222900	200	4231200	100	4221200	100
36004025	4229100	100	4222900	200	4231100	100	4221200	100	4231200	100	4219100	400

Table 3:
Order event data generated by the LOB Constructor.

The sample contains a five second segment of order event (affecting the three-level LOB) data for ticker GOOG on July 1st 2009. The time variable is in milliseconds after the midnight. The type variable indicates the event type: submission, cancellation, execution of a limit order or execution of a hidden orders. Order ID is the ID of the corresponding limit or hidden order. Size is the effective size, i.e. order size for submission events and cancelled (executed) quantity for cancellation (execution) events. The three-level LOB instances immediately after these order events are shown in Table 2.

Time	Type	Order ID	Size	Price	Trade Direction
36000043	1	35859474	100	4231100	-1
36000044	1	35859503	200	4231200	-1
36000207	1	35862501	100	4229100	-1
36000208	3	35859503	200	4231200	-1
36000208	3	35603811	100	4231200	-1
36003222	1	35926475	100	4231200	-1
36003471	3	35293758	100	4223000	1
36004005	1	35948533	100	4229100	-1
36004009	4	35332615	100	4227300	1
36004009	4	35643198	200	4227300	1
36004009	5	35643169	200	4227300	1
36004010	1	35948820	100	4222900	1
36004010	1	35948851	200	4227300	1
36004011	3	35948533	100	4229100	-1
36004015	1	35949144	100	4223300	1
36004016	3	35949144	100	4223300	1
36004017	4	35948851	200	4227300	1
36004018	1	35949411	200	4222800	1
36004018	1	35949425	100	4229100	-1
36004018	1	35949469	200	4227300	1
36004020	3	35949411	200	4222800	1
36004020	3	35949425	100	4229100	-1
36004021	1	35949745	100	4223300	1
36004025	4	35949469	200	4227300	1

- `price`: the price of the limit order or the corresponding executed hidden order (in 0.01 of a cent).
- `trade direction`: the trade direction of the corresponding limit (hidden) order; 1 for buy limit order and -1 for sell limit order. Note that in case of order execution, 1 corresponds to seller-initiated trade (i.e. execution against buy limit order) while -1 corresponds to buyer-initiated trade (i.e. execution against sell limit order).

Both files can be easily loaded and used by analytical software, such as Matlab and R. Because the number of output order events is identical to the number rows in the LOB (i.e. both files have the same number of rows), the two output files can be easily merged. The following is an example of R code for loading and merging the data set.

An Example of R Code Loading LOB Data

```
######### load data ##########################
# load order book data
dataOB <- read.csv("GOOG_20090701_orderbook_3.csv")
# load message data
dataM <- read.csv("GOOG_20090701_message_3.csv")
# merge two datasets
data <- cbind(dataM,dataOB[,-1])
######## load completed #####################

# compute the number of order book levels
nlevels <- (dim(dataOB)[2] - 1)/4
# name the columns
colnms <- c("Time","Type","OrderID","Size","Price","TradeDirection")
for (i in 1:nlevels)
{ colnms <- c(colnms,paste("ASKp",i,sep=""),paste("ASKv",i,sep=""),
        paste("BIDp",i,sep=""),paste("BIDv",i,sep="")) }
colnames(data) <- colnms

# clean up
rm('dataOB','dataM')
```

4.4 Application: Visualization of LOB and Order Flow

The purpose of the LOB and order flow visualization is to help researchers intuitively understand the basic principles of order-driven trading by showing a sequence of changes in the LOB associated with the incoming order events. The upper left box in Figure 7 contains basic information about the displayed ticker and current time, as well as a comment field, which interprets the situation at the given point in time, thus helping to understand the principles better. The box labeled "Order Flow" shows several orders around the current order (violet color), as recorded in the corresponding output file. The box Limit Order Book (Table) box contains rows from the LOB that correspond to the orders displayed in the Order Flow box. The

Description

TICKER = APPLE, TIME = 10:00:50.036

The Order with type Hidden Execution and trade direction Sell at price 143.74$ has arrived.

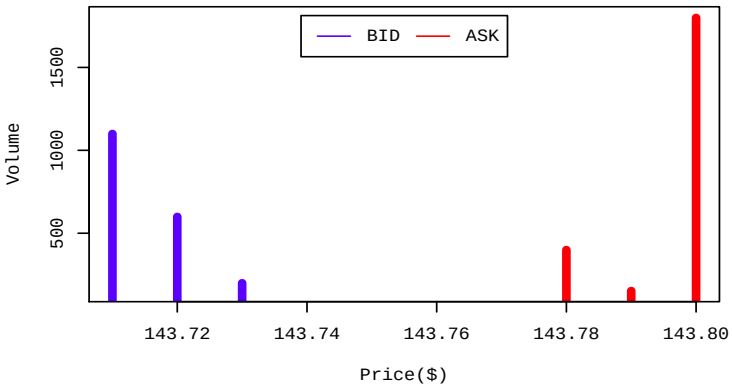
Order Flow

Time	Type	OrderID	Size	Price	Trade
10:00:50.032	Submission	37315193	100	143.78	Sell
10:00:50.036	Visible Exec	37280582	50	143.74	Buy
10:00:50.036	Hidden Exec	37315120	50	143.74	Buy
10:00:50.052	Deletion	37315130	100	143.78	Sell

Limit Order Book (Table)

BID PRICE(BID SIZE)			ASK PRICE(ASK SIZE)		
BLevel 3	BLevel 2	BLevel 1	ALevel 1	ALevel 2	ALevel 3
143.72(600)	143.73(200)	143.74(50)	143.78(400)	143.79(153)	143.8(1800)
143.71(1100)	143.72(600)	143.73(200)	143.78(400)	143.79(153)	143.8(1800)
143.71(1100)	143.72(600)	143.73(200)	143.78(400)	143.79(153)	143.8(1800)
143.71(1100)	143.72(600)	143.73(200)	143.78(300)	143.79(153)	143.8(1800)

Limit Order Book (Graph)



Snapshot of Order flow and LOB at 10:00:50.036 for 3 levels

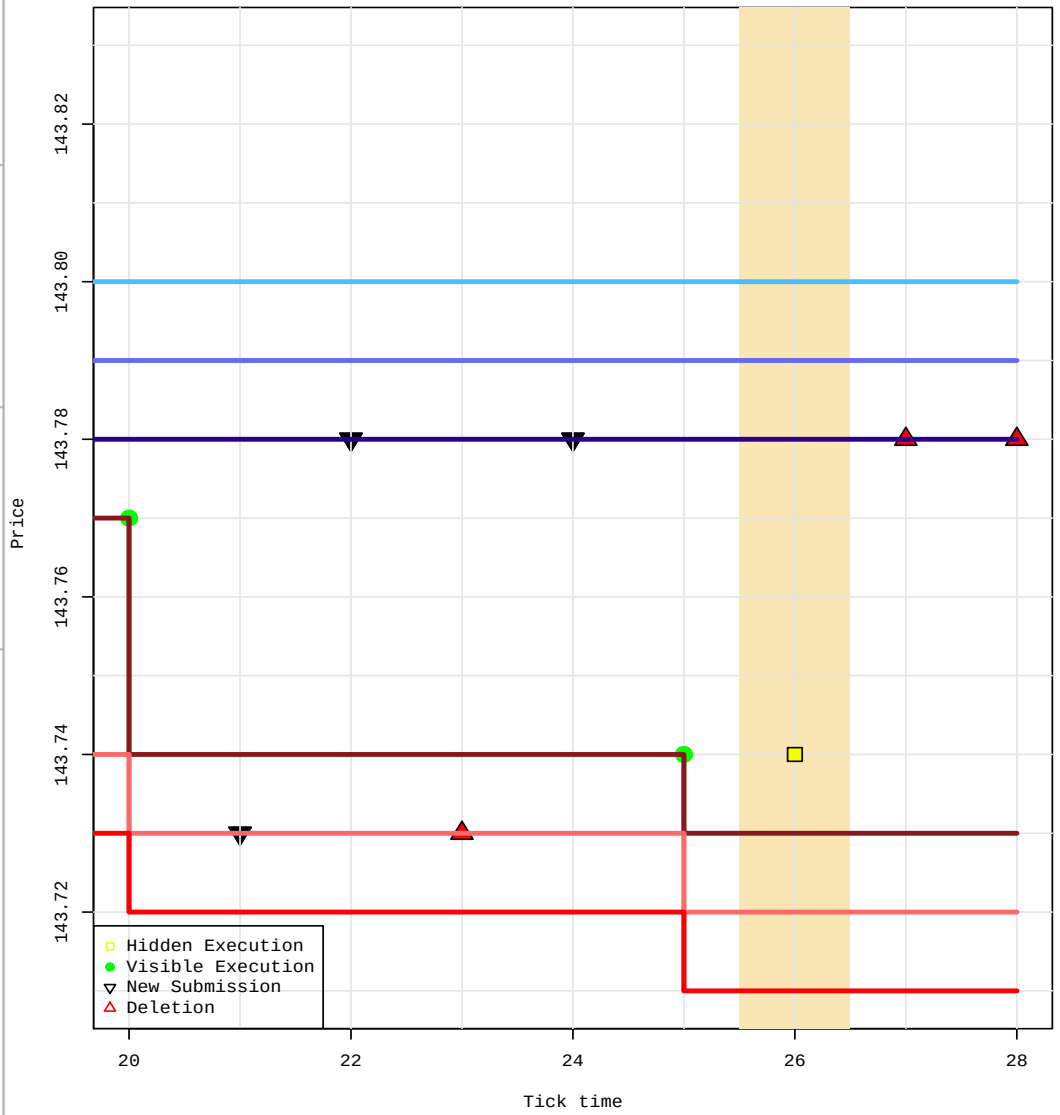


Figure 7: The visualization of LOB and order flow. The time X-axis in the right graph does not correspond to the real time, but rather to "tick time", i.e. events in this graph are equidistant with respect to the X-axis regardless of their actual distance in time. The Y-axis "Price" is scaled realistically to represent the actual quotes in the LOB. The current order event and LOB instance are highlighted by violet color in the left graph and by orange color in the right graph.

Limit Order Book (Table) box contains prices and depth (number of shares) at three levels of bid and ask. The box with the title "Limit Order Book (Graph)" contains the same information (for one row) visualized; depth (number of shares) at the three levels of displayed order book. Note that this graph contains only the current state of the LOB corresponding to the violet row in the previous two boxes. Finally, the large graph on the right displays the incoming orders and the continuity of BID and ASK price levels against the time axis. Note that this graph does not contain information about the size of incoming orders nor the cumulated depth. But in combination, both graphs contain all information from the tables and thus provide current snapshots, as well as the dynamics of the LOB in the sample period.

5 Order Tracer

We also implement a module called `Order Tracer` to extract information, i.e. cancellation, execution and deletion, for single limit orders. The procedure is similar to the `LOB Constructor`.

5.1 Implementation of Order Tracer

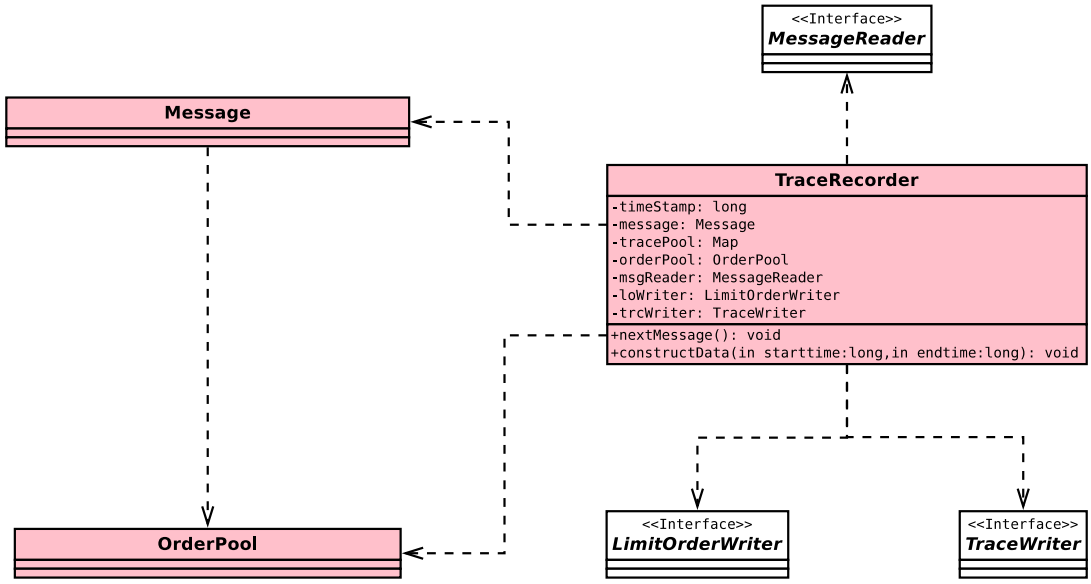


Figure 8: Class diagram of trace construction. A `TraceRecorder` object contains unique `OrderPool` objects, which are updated by incoming messages corresponding to the `Message` objects. Also, it must include at least one `MessageReader` object for reading the input, a `LimitOrderWriter` object and a `TraceWriter` objects for writing the output into the file system.

Reusing the classes designed for the `LOB Constructor`, we implement another application by adding one additional controlling class - the `TraceRecorder`. It passes the message data and completes information from the underlying limit orders exactly in the same way as the `LOB Constructor`. However, rather than using the information to update the LOB, it simply records the types of events and limit orders temporarily in a block of memory called `tracePool`. The

constructed data are then saved by instances of `LimitOrderWriter` and `TraceWriter`. Figure 8 contains our final class diagram.

5.2 Output of Order Tracer

Table 4:
Order event data generated by the Order Tracer.

The sample contains a segment of data for GOOG from October 1st 2010. The time variable is in milliseconds after midnight. Size is the effective size, i.e. cancelled (executed) quantity of cancellation (execution) events. Variable Execution indicates the type of event (execution or cancellation). Variable Earlier exe. indicates the first execution by zero. The underlying limit order information is shown in Table 5.

Submission Time	Time	Size	Execution	Earlier exe.
36502739	36502742	100	0	0
36502743	36502743	100	0	0
36502554	36502744	100	0	0
36502744	36502745	100	0	0
36502726	36502745	100	0	0
36502734	36502751	100	0	0
36502745	36502755	100	1	0
36502742	36502755	123	1	0
36502755	36502755	100	1	0
36502745	36502756	100	0	0
36502730	36502756	100	0	0
36502742	36502757	177	1	1
36502750	36502757	100	1	0
36502741	36502757	23	1	0
36502741	36502757	77	1	1
36502737	36502757	100	1	0
36502047	36502757	100	0	0
36502761	36502761	100	1	0
36502757	36502761	300	0	0
36502761	36502761	100	0	0

Similar to the `LOB Constructor`, the `Order Tracer` generates two files with the same number of rows. The first output file classifies events as cancellations or executions and contains the following six columns (see a sample in Table 4).

- `submission time`: the time when the corresponding limit order was submitted; milliseconds after midnight.
- `time`: time stamp of the event; milliseconds after midnight.
- `size`: change of limit order size; number of shares.

- `execution`: a dummy variable indicating whether the event corresponds to a limit order execution (1) or cancellation (0).
- `earlier exe.`: a dummy variable indicating whether the event corresponds to a limit order that has been partially executed earlier (1) or not (0).

Table 5:
Limit order data generated by Order Tracer.

The sample contains a segment of data for GOOG from October 1st 2010. Variable Rem. Size is the remaining share quantity of the limit order after an event. The Price variable is in 0.01 cent. Some limit orders may contain additional MPID information identifying the submitters. Events recorded for individual limit orders are shown on Table 4.

Order ID	Order Size	Rem. Size	Price	Trade Dir.	Hidden	MPID
57916505	100	0	5273000	1	0	null
57916618	100	0	5273100	1	0	null
57913456	100	0	5261200	1	0	HDSN
57916642	100	0	5273300	1	0	null
57916188	100	0	5271700	1	0	null
57916358	100	0	5271700	1	0	null
57916698	100	0	5273400	1	0	null
57916588	300	177	5273300	1	0	null
57916928	100	0	5273400	1	0	null
57916679	100	0	5273000	1	0	null
57916279	100	0	5221600	1	0	NMRA
57916588	300	0	5273300	1	0	null
57916814	100	0	5273300	1	0	null
57916545	100	77	5273100	1	0	null
57916545	100	0	5273100	1	0	null
57916448	100	0	5273000	1	0	null
57899910	100	0	5275100	-1	0	null
57917068	100	0	5273300	1	0	null
57917003	300	0	5265700	1	0	null
57917079	100	0	5275000	-1	0	null

The second output file contains information about the corresponding limit orders. Table 5 contains a segment of this file. There are seven variables:

- `order ID`: a unique number generated by the exchange identifying the limit order.
- `order size`: original order size at submission.
- `remaining size`: remaining size of the order after an order event.
- `price`: price of the limit order.
- `trade direction`: 1 for buy limit orders, -1 for sell limit orders.

- `hidden`: a dummy variable indicating whether the order is hidden (1) or not (0).
- `MPID`: Market Participant ID of the trader who submitted the order, null if unknown.

Since we organize the constructed data in such a way that the two output files contain identical number of observations, we can easily load and merge them using statistical software, such as Matlab and R. Here is an example of Matlab code.

An Example of Matlab Code Loading Order Trace Data

```
% load event data
trace_data=load('GOOG_20101001_trace.csv']);

% load order data
fid=fopen(['GOOG_20101001_order.csv']);
orders=textscan(fid,'%f,%f,%f,%f,%f,%s');
fclose(fid);
order_data=[orders{1} orders{2} orders{3} orders{4} orders{5} orders{6}];

% merge the datasets
mydata=[traceData orderData];

% clean up
clear trace_data order_data orders;
```

5.3 Application: Main Characteristics of Limit Orders

Figure 9 shows a simple analysis of the characteristics of limit orders using the output of the `Order Tracer`. Calculation of the order sizes, execution quantities and life time of limit orders are trivial tasks when using the data illustrated in Table 4 and 5. Nevertheless, we can make a few interesting empirical observations: 1) Market participants submit a huge number of limit orders with small sizes. Indeed, we find that most limit orders are of size 100, which is the size of a round lot in NASDAQ. 2) Only a small proportion of limit orders are executed and the execution quantity is small. 3) Most of limit orders are cancelled shortly after the submission. 4) Execution time is typically longer than cancellation time. More detailed analysis of order flow properties and limit order characteristics can be found in Hautsch and Huang (2011a).

6 Conclusion

System LOBSTER is designed to meet the three basic requirements for constructing datasets for empirical studies on limit order markets. First, it is sufficiently efficient and *fast*, requiring only seconds or minutes to fulfill standard requests. The system is web-based and very intuitive with a *user-friendly* interface allowing researchers to fully focus on research rather than spend time preparing data. Third, the system was programmed using object-oriented programming

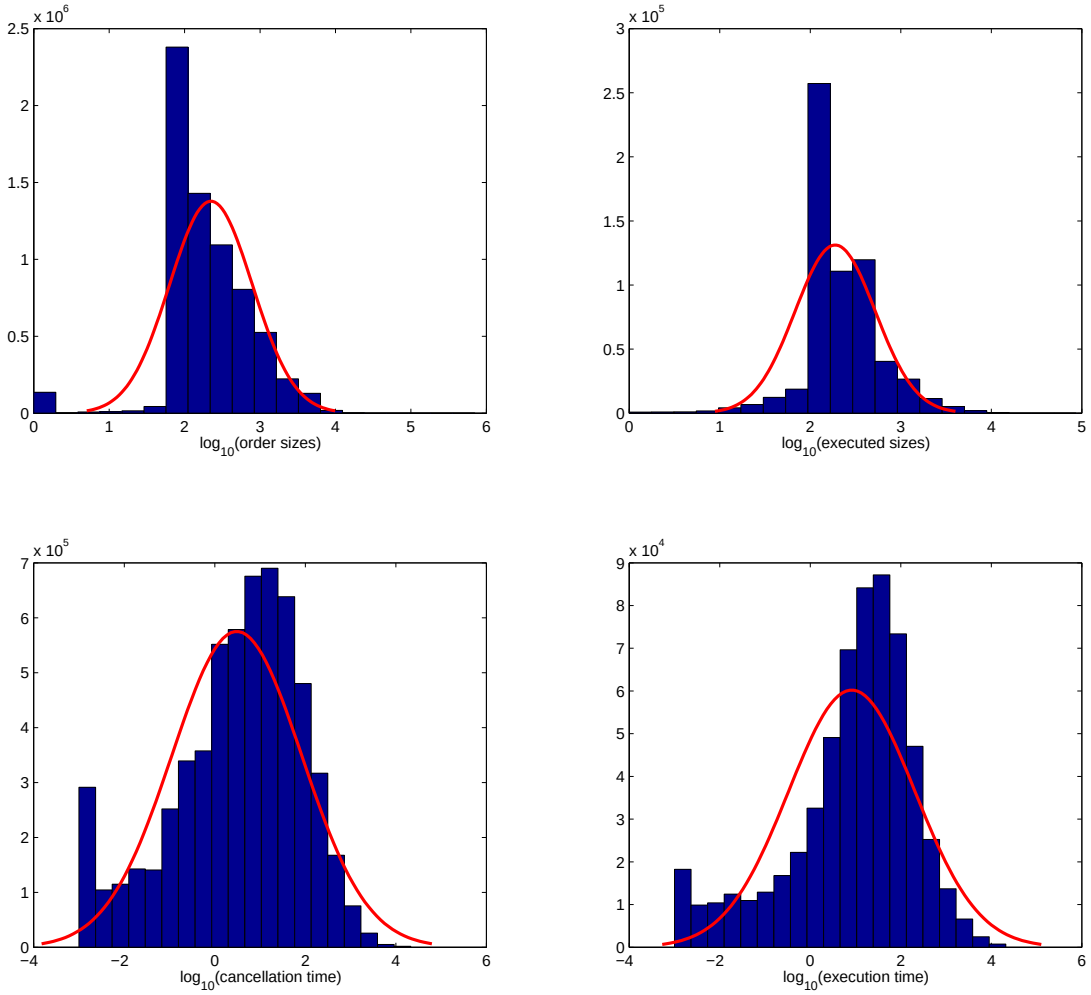


Figure 9: Histogram of size, execution quantity, cancellation time and execution time for limit orders. The red line represents kernel density estimates. Zero cancellation time and execution are discarded. Trading data for Microsoft Corp. on NASDAQ in October, 2010

language and intentionally designed to allow for easy extensions, which makes it very *versatile*. New modules of the LOB Constructor and Order Tracer have been already added to the system.

Of course, any meaningful extension must be based on a sound research idea. To facilitate communication with other researchers, we created a forum (`http://lobster.wiwi.hu-berlin.de/forum/`) focused on modelling order book and order flow. It is a slowly but steadily growing source of information, not only about LOBSTER, but about everything related to order-driven markets. We expect that it will become a rich and comprehensive pool of references for academic researchers, which will help them to accelerate the initial stages of their research projects and help us further develop the system. The feedback we are receiving from students and researchers pioneering with our system has already proven to be essential in the development of the system.

References

Hasbrouck, J., and G. Saar, 2009, Technology and liquidity provision: The blurring of traditional definitions, *Journal of Financial Markets* 12, 143 – 172.

Hautsch, N., and R. Huang, 2011a, Limit order flow, market impact and optimal order sizes: Evidence from NASDAQ TotalView-ITCH data, Discussion Paper 2011-056 Sonderforschungsbereich 649, Humboldt Universität zu Berlin, Germany.

——— , 2011b, On the dark side of the market: Identifying and analyzing hidden order placement, Discussion paper Humboldt Universität zu Berlin, Germany.