TAC Market Design: Communication Protocol Specification

Version 1.04
October 6, 2006

Contents

1 Introduction **3**

2 Change Log **3**

3 Game Configuration **3**
3.1 Timing . . . 3
3.2 Trading Agents . . . 4
3.3 Market Selection . . . 5
3.4 Fees . . . 5
3.5 Evaluation . . . 5

4 Notational Conventions and Generic Grammar **6**

5 Protocol Parameters **6**
5.1 CATP Version . . . 6

6 Connections **7**

7 CATP Messages **7**
7.1 Client ID . . . 8
7.2 Shout ID . . . 8
7.3 Transaction ID . . . 8
7.4 Message Types . . . 8
7.5 Message Headers . . . 9
7.6 Message Body . . . 10

8 Requests **10**
8.1 CHECKIN 10
8.2 REGISTER 11
8.3 SUBSCRIBE 11
8.4 ASK 12
8.5 BID 14
8.6 TRANSACTION 14
8.7 GET 15
8.8 POST 16
8.9 OPTIONS 18
8.9.1 GAMESTART 18
8.9.2 GAMEOVER 18
8.9.3 DAYOPEN 18
8.9.4 DAYCLOSE 19
8.9.5 ROUNDOPEN 19
8.9.6 ROUNDCLOSE 20

9 Responses **20**
9.1 OK 20
9.2 INVALID 21
9.3 ERROR 21

10 Message Headers **21**
10.1 Id 21
10.2 Type 22
10.3 Value 22
10.4 Text 22

11 Possible catp Dialogs **22**
11.1 Interactions during a CAT game 22
11.2 Interactions during a CAT game: Seller's dialog 22
11.3 Interactions during a CAT game: Buyer's dialog 22
11.4 GET messages 26

1 Introduction

This document presents the specification of communication protocol for TAC Market Design Competition, or CAT, games. The protocol is named the CAT Protocol, or in short CATP.

2 Change Log

- Version 1.00 : first version by Jinzhong Niu in July, 2006.
- Version 1.01 : (Jinzhong Niu, 08/02/2006) additional diagrams by Albert Mmoloke showing possible interactions during a CAT game added in Section 11.
- Version 1.02 : (Jinzhong Niu, 08/06/2006) typos and inconsistencies fixed.
- Version 1.03 : (Jinzhong Niu, 08/28/2006) HELLO and MATCH messages renamed to CHECKIN and TRANSACTION; SUBSCRIBE message added.
- Version 1.04 : (Simon Parsons, 10/05/2006) typographical fixes, some additonal explanation added.

3 Game Configuration

This section introduces the configuration of CAT games, helping make this document self-explanatory. For more information, please refer to [1].

It is assumed that a CAT game consists of a CATP server and several CATP clients, which may be trading agents or specialists (markets). CATP clients do not talk to each other directly; instead they connect to the CATP server through sockets and the server responds to messages from clients and forward information if needed.

NOTE: The meta-exchange in the CAT game architecture in [1] is not taken into consideration in this specification for now.

3.1 Timing

A CAT game lasts a certain number of *day*s and each day consists of *round*s. The CATP server in a game has the control on a central clock and notifies clients of the following events:

- start of game
- end of game
- start of a day (day open)
- close of a day
- start of a round (round open)
- close of a round

In order to allow traders and specialists to exploit some deadline effects, the server also notifies clients of the length of a day — in term of number of rounds — and the length of each round — in term of the number of physical seconds — before a game starts.

NOTE: As described below, the length of a game — in terms of the number of days — is not disclosed by the server to any of the clients.

3.2 Trading Agents

In a CAT competition, trading agents are provided by the CAT game, each trading agent is associated with a trading strategy and has a private value for the good being traded.

The set of trading strategies that may be used by traders in CAT games include:

- GD [3]
- ZIP [2]
- RE (Roth and Erev) [6]
- ZI-C [4]

Private values, which determine the demand and supply of the market, are allocated by the CATP server, and change from day to day. However, private values remain constant during a day.

NOTE: The sizes of the sub-populations of trading agents adopting a different trading strategy may or may not be made public before a game takes place.

3.3 Market Selection

Traders transact business through specialists. When a trading day opens, each trader registers with one specialist (and only one specialist) and stays with that specialist until the day closes. Traders may move to other specialists over days. A trader is only allowed to make shouts and transactions through the specialist it registers with.

NOTE: The full set of strategies that traders will use in order to select between specialists have yet to be developed and determined, and it is still under discussion whether they should be made public or not in advance.

3.4 Fees

A specialist can makes a profit by charging a trader when the latter:

- registers with the specialist;
- requests to receive information on shouts and/or transactions taking place through the specialist;
- makes a shout; and
- is involved in a transaction.

A specialist is free to set the level of these charges (from zero up).

Specialists are requireed to send information to the server on how much they charge for each type of activity when a trading day opens. Specialists are permitted to adapt their charging policies over time.

3.5 Evaluation

Specialists in a CAT game will be evaluated across a number of performance metrics. One such metric is the profit they make through the game. This profit is the sum of all the fees the specialist receives, plus the sum of all sales the specialist makes, minus the cost of all the purchases the specialist makes. If a specialist pays to obtain information on shouts and transactions taking place in markets run by other specialists, the charges incurred are deducted from its profit.

NOTE: The current version of the CAT game *only* evaluates specialists by the profit that they make.

4 Notational Conventions and Generic Grammar

The following rules from the HTTP 1.1 Specification [5] are used throughout this specification to describe basic parsing constructs.

```
OCTET          = <any 8-bit sequence of data>
CHAR           = <any US-ASCII character (octets 0 - 127)>
UPALPHA        = <any US-ASCII uppercase letter "A".."Z">
LOALPHA        = <any US-ASCII lowercase letter "a".."z">
ALPHA          = UPALPHA | LOALPHA
DIGIT          = <any US-ASCII digit "0".."9">
CTL            = <any US-ASCII control character
                (octets 0 - 31) and DEL (127)>
CR             = <US-ASCII CR, carriage return (13)>
LF             = <US-ASCII LF, linefeed (10)>
SP             = <US-ASCII SP, space (32)>
HT             = <US-ASCII HT, horizontal-tab (9)>
<">            = <US-ASCII double-quote mark (34)>

CRLF           = CR LF

LWS            = [CRLF] 1*( SP | HT )

TEXT           = <any OCTET except CTLs,
                 but including LWS>

HEX            = "A" | "B" | "C" | "D" | "E" | "F"
               | "a" | "b" | "c" | "d" | "e" | "f" | DIGIT

token          = 1*<any CHAR except CTLs or separators>
separators     = "(" | ")" | "<" | ">" | "@"
               | "," | ";" | ":" | "\" | <">
               | "/" | "[" | "]" | "?" | "="
               | "{" | "}" | SP | HT
```

5 Protocol Parameters

5.1 CATP Version

CATP uses a `"<major>.<minor>"` numbering scheme to indicate versions of the protocol. The protocol versioning policy is intended to allow the sender to indicate the format of a message and its capacity for understanding further CATP communication, rather than the features obtained via that communication. No change is made to the version number for the addition of message components which do not affect communication behavior or which only add to extensible field values. The `<minor>` number is incremented when the changes made to the protocol add features which do not change the general message parsing algorithm, but which may add to the message semantics and imply additional capabilities of the sender. The `<major>` number is incremented when the format of a message within the protocol is changed.

```
CATP-Version   = "CATP" "/" 1*DIGIT "." 1*DIGIT
```

NOTE: The major and minor numbers must be treated as separate integers and that each may be incremented higher than a single digit.
See Section 8.1 for how CATP version is used in `CHECKIN` messages.

6 Connections

A CATP connection is setup between a CATP server and a CATP client for message transmission.

A server listens on port `9090`[1] upon startup. Clients connect to the port and establish connections. CATP connections work in a way that is similar to a persistent HTTP connection. Normally they should be kept alive from the beginning of a game through the end.

In case a connection is broken while a competition is running, the client involved may attempt to establish a new connection to the server and take part in the game beginning with the next trading day. It is the responsibility of the client to detect that a connection has been broken, and to establish a new connection.

7 CATP Messages

In a CAT game, plain-text messages are transmitted over CATP connections for the CATP server and CATP clients to communicate with each other.

[1]The port number could be chosen arbitrarily as long as it does not conflict with any existing known service.

Since a message is always transmitted over a point-to-point CATP connection, there is no need to include the identities of the sender and the receiver in the message.

It is the server's responsibility to keep copies of messages from clients in some way for the purposes of security, robustness, and game-wide information integrity[2].

7.1 Client id

Each client, trader or specialist, in a CAT game is assigned a unique ID by the server, which is used in messages whenever needed to specify a client. A client ID should be a `token`.

```
client-id   = token
```

7.2 Shout id

After a shout is placed by a trader and confirmed as valid by the server, it is assigned a unique ID by the server, which has identical syntax to client IDs.

```
shout-id   = token
```

7.3 Transaction id

After a transaction is made by a specialist and confirmed valid by the server, it is assigned a unique ID by the server, which has identical syntax to client IDs.

```
transaction-id   = token
```

7.4 Message Types

Following the convention with HTTP messages, CATP messages include requests and responses.

NOTE: Requests can be made by either a client or server in CATP, which is not true however in HTTP.

```
message   = request | response
```

[2][1] proposes a stand-alone registry component for logging all the messages, but discussion of it is beyond the scope of this document since its existence and behavior do not affect the way CATP is designed and implemented.

Both types of message consist of a `start-line`, zero or more header fields (also known as `headers`), an empty line (i.e., a line with nothing preceding the `CRLF`) indicating the end of the header fields, and possibly a `message-body`.

```
generic-message = start-line
                  *(message-header CRLF)
                  CRLF
                  [ message-body ]
start-line      = message-type CRLF
message-type    = request-type
                | response-type
```

Often, the `message-type` is sufficient to completely identify the subject of a message. When this is not the case, a `Type` header should be used to provide additional information (see Sections 8.1, 8.4, 8.5, 8.7, 8.9, 8.8 and 10.2 for more information).

7.5 Message Headers

CATP message header fields are included in messages providing additional information.

Each header field consists of a name followed by a colon (`:`) and the field value. Field names are case-insensitive. The field value may be preceded by any amount of `LWS`, though a single `SP` is preferred.

```
message-header = field-name ":" [ field-value ]
field-name     = token
field-value    = *( field-content | LWS )
field-content  = <the OCTETs making up the field-value
                 and consisting of either *TEXT or
                 combinations of token, separators,
                 and quoted-string>
```

The `field-content` does not include any leading or trailing `LWS` occurring before the first non-whitespace character of the `field-value` or after the last non-whitespace character of the `field-value`. Such leading or trailing `LWS` may be removed without changing the semantics of the field value. Any `LWS` that occurs between `field-content` may be replaced with a single `SP` before interpreting the field value or forwarding the message downstream.

The order in which header fields with differing field names are received is not significant.

Multiple `message-header` fields with the same `field-name` may be present in a message if and only if the entire `field-value` for that header field is defined as a comma-separated list, i.e. `#(value)`. It must be possible to combine the multiple header fields into one `field-name: field-value` pair, without changing the semantics of the message, by appending each subsequent `field-value` to the first, each separated by a comma.

7.6 Message Body

No CATP message currently uses a message body.

8 Requests

A request message requests the receiver to take actions. Types of request include:

```
request-type   = "CHECKIN"
               | "REGISTER"
               | "SUBSCRIBE"
               | "ASK"
               | "BID"
               | "TRANSACTION"
               | "GET"
               | "POST"
               | "OPTIONS"
```

8.1 CHECKIN

A `CHECKIN` request should be sent by a CATP client once it connects to the server, including as a request parameter the highest version of CATP the client can support.

A `Type` header and a `Text` header are required in a `CHECKIN` request to provide the type and human-readable name of the CATP client.

EXAMPLE:

```
CHECKIN CATP/1.0 CRLF
Type: Seller CRLF
Text: CUNY Bot 1.0 CRLF
CRLF
```

On receiving a `CHECKIN` request, the server allocates a client ID and sends back an `OK` response.

EXAMPLE:

```
OK CRLF
Id: Seller5 CRLF
CRLF
```

8.2 REGISTER

A `REGISTER` request from a trader client to the server expresses its desire to trade through the specialist specified in an `Id` header.

EXAMPLE:

```
REGISTER CRLF
Id: specialist3 CRLF
CRLF
```

If the registration is successful, the server responds with an empty `OK` message.

EXAMPLE:

```
OK CRLF
CRLF
```

NOTE: The server knows which trader registers with which specialist, and checks the validity of shouts and transactions based on its records[3]. Presently, specialists have no knowledge about registered traders, which though may be changed later on to allow specialists to adapt based on the identity of traders.

8.3 SUBSCRIBE

A `SUBSCRIBE` request from a trader to the server tells the latter the trader would pay for information regarding shouts and transactions made at some specialists. A `ID` header gives a list of specialists from which the trader want to purchase information.

EXAMPLE:

[3]All messages received by the CAT server are logged in the registry

```
SUBSCRIBE CRLF
ID: specialist1, specialist3 CRLF
CRLF
```

The server responds to a `SUBSCRIBE` message with a `OK` message and then notifies those specialists by sending a `SUBSCRIBE` message with an `ID` header telling which trader subscribes.

EXAMPLE:

```
SUBSCRIBE CRLF
ID: trader2 CRLF
CRLF
```

A specialist replies to a `SUBSCRIBE` with a `OK` and may use the notification to calculate its profit for its personal use (for example feeding the signal into some learning algorithm to adjust its parameters).

`SUBSCRIBE` requests can be sent any time during a trading day and the server is responsible for broadcasting the necessary information to subscribed traders.

8.4 ASK

An `ASK` request from a seller to the server expresses an offer to sell through the specialist the seller is registered with. A `Value` header is required to hold the ask price.

EXAMPLE:

```
ASK CRLF
Value: 87 CRLF
CRLF
```

After an ask arrives at the server,

- if the ask fails in validation, the server simply sends an `INVALID` response.

  EXAMPLE:

  ```
  INVALID CRLF
  CRLF
  ```

- Otherwise, the server allocates a unique ID to the ask and sends the ID back to the trader in an `OK` response.

  EXAMPLE:

  ```
  OK CRLF
  Id: ask2 CRLF
  CRLF
  ```

  Further, the original `ASK` message should be forwarded to the specialist with an additional `Id` header to include the ask ID.

  EXAMPLE:

  ```
  ASK CRLF
  Id: ask2 CRLF
  Value: 87 CRLF
  CRLF
  ```

After an `ASK` message is received by a specialist, it is accepted or rejected according to the rules of the specialist. For example, if the specialist requires that every ask beat the standing ask, then an ask that is too high will be rejected.

- If accepted, an empty `OK` response should be sent to the server.

  The server then notifies the seller and possibly other traders if they paid for information on shouts placed. The message from the server is identical to the one above sent by the server to the specialist. The receivers then simply respond with an empty `OK` message to confirm.

- If rejected, an `INVALID` response should be created and sent to the server with optional `Type` and `Text` headers giving the type of reason and detailed description.

  EXAMPLE:

  ```
  INVALID CRLF
  CRLF
  ```

The server then sends an `ASK` message to the seller with `INVALID` in a `Type` header and the ask ID in an `Id` header. Since the ask is not accepted, only the seller ought to be notified in this case. Again, the seller should send back an empty `OK` message to confirm.

Please refer to Figure 2 for illustrations of how asks are processed.

NOTE: A specialist does not know which seller made an ask according to the above protocol. This can be changed in later versions of CATP though.

8.5 BID

A `BID` sent from a buyer to the server expresses an offer to buy through a specialist. The processing of `BID` messages is similar to that of `ASK` messages.

8.6 TRANSACTION

A `TRANSACTION` request, made by a specialist to the server, announces a transaction made by the specialist. An `Id` header should be used to provide the corresponding ask ID and bid ID in order and a `Value` header for the transaction price.

EXAMPLE:

```
TRANSACTION CRLF
Id: ask3, bid2 CRLF
Value: 90 CRLF
CRLF
```

The server checks whether this transaction is valid or not.

- If the transaction is valid, the server should allocate a unique transaction ID for the transaction and respond with an `OK` message with the ID in an `Id` header.

  EXAMPLE:

  ```
  OK CRLF
  Id: transaction2 CRLF
  CRLF
  ```

  The server further sends a `TRANSACTION` message to both the buyer and the seller involved in the transaction as well as other traders who paid for the information about the transaction. The message should include the transaction ID, the ask ID, and the bid ID in order in an `Id` header and the transaction price in a `Value` header.

EXAMPLE:

```
TRANSACTION CRLF
Id: transaction2, ask3, bid2 CRLF
Value: 90 CRLF
CRLF
```

Traders receiving this message successfully should respond with an empty `OK` message.

- If the transaction is invalid for example because the price in the ask, as logged in the registry, is higher than the price in the bid, also logged in the registry, the server should send back an `INVALID` message, with optional `Type` header and a `Text` header giving respectively the reason of the message being invalid and detailed description.

Please refer to Figure **??** for illustrations of processing transactions.

NOTE: A transaction cannot be rejected by the traders involved in a transaction. Its validity is guaranteed by the CATP server. This makes the CAT markets different to some (for example that in the Santa Fe auction tournament [7]) which allows traders to decide whether to accept matching offers.

8.7 GET

A `GET` request from a CATP client to the server asks for information that is accessible without charge. A `Type` header should be used to specify what type of information is requested. It can be

- `FEE`: fees charged by a specialist, i.e. a price list. The specialist ID should be specified in an `Id` header.

EXAMPLE:

```
GET CRLF
Type: FEE CRLF
Id: Specialist3 CRLF
CRLF
```

A positive response from the server should include a `Value` header presenting the price list with comma as separator and optionally the specialist ID.

There are different kinds of fees charged by a specialist, these include:

- a fee for traders registering with a market
- a fee for traders or specialists receiving information on shouts and transactions
- a fee for traders making a shout
- a fee for a shout being matched by a specialist

These fees should be listed in the above order in the `Value` header as numeric values.

EXAMPLE:

```
OK CRLF
Id: Specialist3 CRLF
Value: 2, 5, 4, 6
CRLF
```

NOTE: Only these charges, and only flat charges are allowed for now. Charges that depend on another value(s), like 5 percent of a transaction price, will be supported later on.

- `TRADER`: list of trader clients. An `Id` header is used by the server in the response given a list of trader client IDs.
- `SPECIALIST`: list of specialist clients. The processing of this type of request is similar to that of `TRADER`.
- `PROFIT`: profit made by the specified specialist (so far). A `Value` header should be included to give the amount of profit.

8.8 POST

`POST` messages are used to proactively provide (push) information to receivers. This is the same information as may be requested (pulled) by `GET` messages.

The following information may be provided through `POST`:

- After a CAT game starts, but before the first day begins, the server is required to send `POST` messages to all the traders and specialists providing the list of specialists, and the list of traders.

  EXAMPLE:

  ```
  POST CRLF
  Type: SPECIALIST CRLF
  Id: Specialist0, Specialist1, Specialist2 CRLF
  CRLF

  POST CRLF
  Type: TRADER CRLF
  Id: Buyer0, Buyer1, Seller0 CRLF
  CRLF
  ```

- Specialists send their price lists to the server before a trading day opens.

- The server sends price lists to traders once a trading day opens.

  EXAMPLE:

  ```
  POST CRLF
  Type: FEE CRLF
  Id: Specialist0 CRLF
  Value: 2, 5, 4, 6 CRLF
  CRLF

  ...

  POST CRLF
  Type: FEE CRLF
  Id: Specialist2 CRLF
  Value: 5, 2, 4, 6 CRLF
  CRLF
  ```

  NOTE: It seems reasonable that specialists should also have access to the pricing policies of other specialists so as to adapt their own charging policies. This will be supported later.

8.9 OPTIONS

OPTIONS requests are control messages sent by the CATP server to clients, involving timing, demand/supply schedule, etc. Types of OPTIONS messages are described in a Type header.

8.9.1 GAMESTART

A GAMESTART OPTIONS message notifies CATP clients of the start of the game. A Value header with 2 integer values should be used along with the GAMESTART to inform traders of the length of a day and the length of a round.

EXAMPLE:

```
OPTIONS CRLF
Type: GAMESTART CRLF
Value: 20, 3600 CRLF
CRLF
```

The GAMESTART OPTIONS message is followed by a POST TRADER message and a POST SPECIALIST message.

NOTE: The length of a game in days will not be known to CATP clients. The reason for this is to prevent specialists exploiting deadline effects due to the end of the game. The reason for this is that the CAT game is about desiging markets with good steady-state behavior, and thus markets that work well on any given day.

8.9.2 GAMEOVER

A GAMEOVER OPTIONS message notifies CATP clients of the end of the game.

EXAMPLE:

```
OPTIONS CRLF
Type: GAMEOVER CRLF
CRLF
```

8.9.3 DAYOPEN

A DAYOPEN OPTIONS message notifies CATP clients of the open of a trading day.

In a `DAYOPEN` `OPTIONS` messages to a trader, a `Value` is used to assign the private value. Multiple values indicate multiple items for the trader to trade. An `OK` response from traders is expected.

EXAMPLE:

```
OPTIONS CRLF
Type: DAYOPEN CRLF
Value: 65 CRLF
CRLF

OK CRLF
CRLF
```

`DAYOPEN` `OPTIONS` messages to specialists do not include extra information, but the responses from specialists should include a price list in a `Value` header.

EXAMPLE:

```
OPTIONS CRLF
Type: DAYOPEN CRLF
CRLF

OK CRLF
Value: 2, 5, 4, 6 CRLF
CRLF
```

8.9.4 DAYCLOSE

A `DAYCLOSE` `OPTIONS` message notifies CATP clients of the close of a trading day.

EXAMPLE:

```
OPTIONS CRLF
Type: DAYCLOSE CRLF
CRLF
```

8.9.5 ROUNDOPEN

A `ROUNDOPEN` `OPTIONS` message notifies CATP clients of the open of a trading round.

EXAMPLE:

```
OPTIONS CRLF
Type: ROUNDOPEN CRLF
CRLF
```

8.9.6 ROUNDCLOSE

A ROUNDCLOSE OPTIONS message notifies CATP clients of the close of a trading round.

EXAMPLE:

```
OPTIONS CRLF
Type: ROUNDCLOSE CRLF
CRLF
```

9 Responses

After a request message is received and interpreted, a CATP response message should be sent back.

The first line of a response message consists of the type of response, and an optional sender and receiver ID pair.

```
response-type   = "OK"
                | "INVALID"
                | "ERROR"
```

9.1 OK

An OK response acknowledges the success of a request. For example, a seller agent may receive a response like the following:

```
OK CRLF
Id: ask3
CRLF
```

informaing it that the ASK request is successful.

9.2 `INVALID`

An `INVALID` response indicates that the corresponding request is invalid, meaning the request is syntax-correct but it is not expected or logically sound — for example when a seller sends out a `BID` request, or a seller sends out an `ASK` request that violates some market rule (like having to beat the current quote).

9.3 `ERROR`

An `ERROR` response notifies the sender of the corresponding request of occurrence of error. The `ERROR` message may need a `Type` header to tells what type of error it is and an optional `Text` header for additional description.

Types of error supported include:

- `REQUEST`: syntax error in request or failure in reading request

  NOTE: It is up to the receiver whether to disconnect from the sender and try to connect again.

- `RESPONSE`: failure while trying to respond

EXAMPLE:

```
ERROR CRLF
Type: REQUEST CRLF
Text: syntax error in request CRLF
CRLF
```

10 Message Headers

This section gives detailed descriptions of CATP header fields, as well as links to related message types.

10.1 `Id`

A `Id` header field provides one or more IDs, separated by commas if necessary. The number and semantics of IDs are determined by the context of the current message. See Sections 8.4, 8.5, 8.6, and 9.1.

10.2 `Type`

A `Type` header field gives the type of an entity. See Sections 8.1, 8.7, 8.8, 8.9, and 9.3.

10.3 `Value`

A `Value` header field provides one or more numeric values. The number(s) and semantics of values depends upon the context of the current message. See Sections 8.4, 8.5, and 8.6.

10.4 `Text`

A `Text` header field carries a plain-text string providing additional description. See Section 8.1, 9.2, and 9.3.

11 Possible catp Dialogs

11.1 Interactions during a cat game

Figure 1 shows the order of message passing among the CATP server and CATP clients during a CAT game.

11.2 Interactions during a cat game: Seller's dialog

Figure 2 shows the order of message passing among the CATP server and CATP sellers during a CAT game.

11.3 Interactions during a cat game: Buyer's dialog

Figure 3 shows the order of message passing among the CATP server and CATP buyers clients during a CAT game.

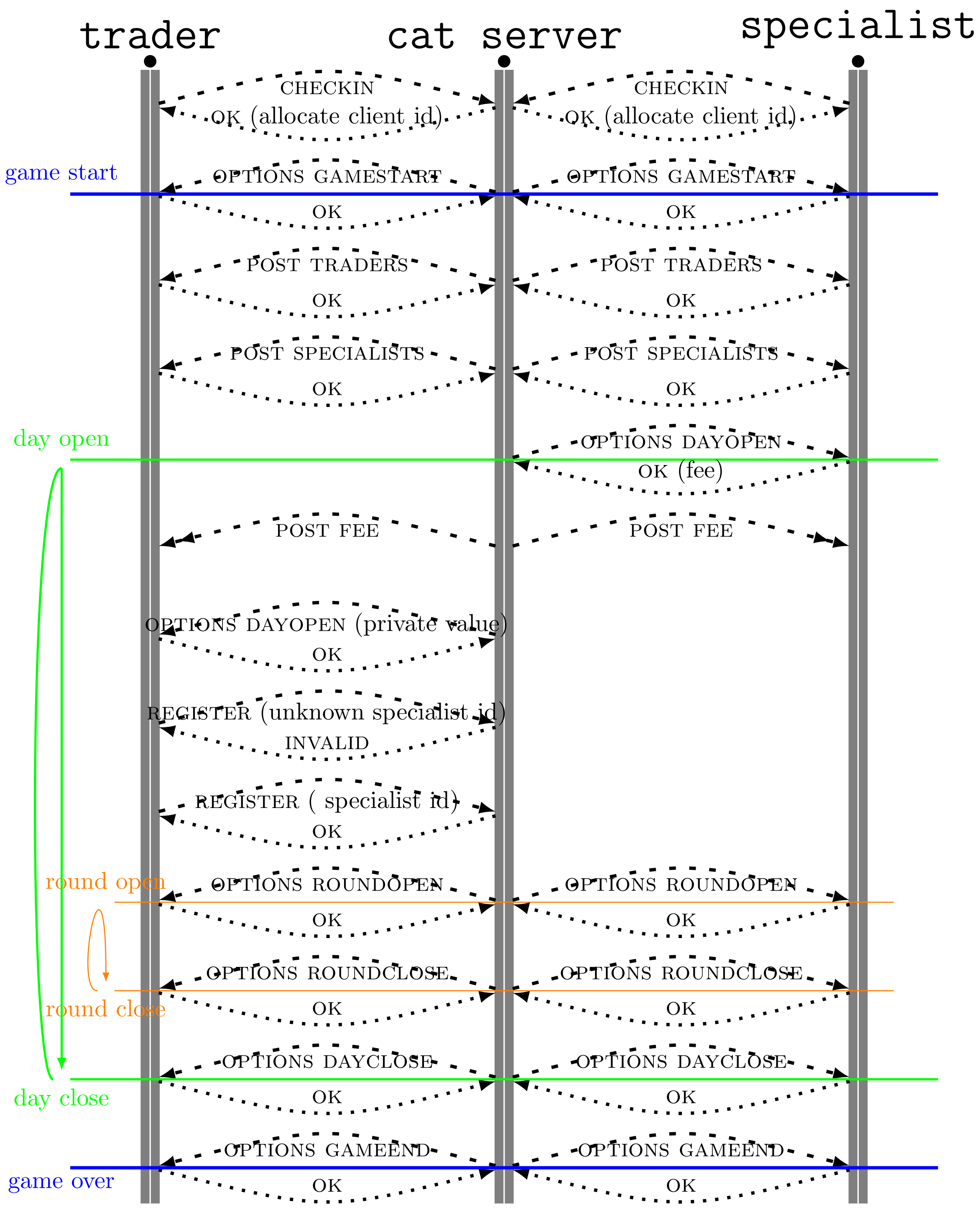

Figure 1: Illustration of interactions during a CAT game

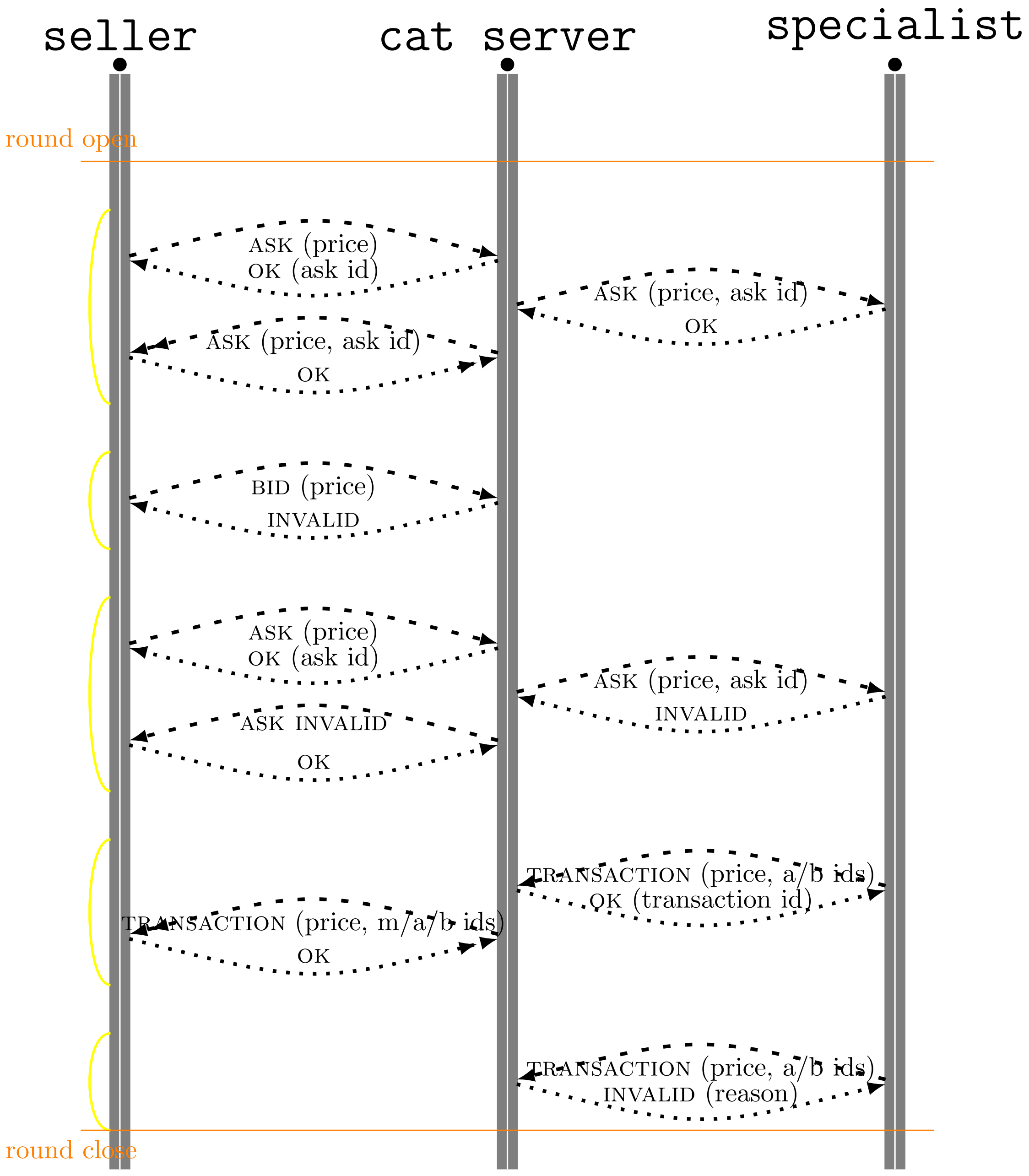

Figure 2: Seller's dialog during a trading round.

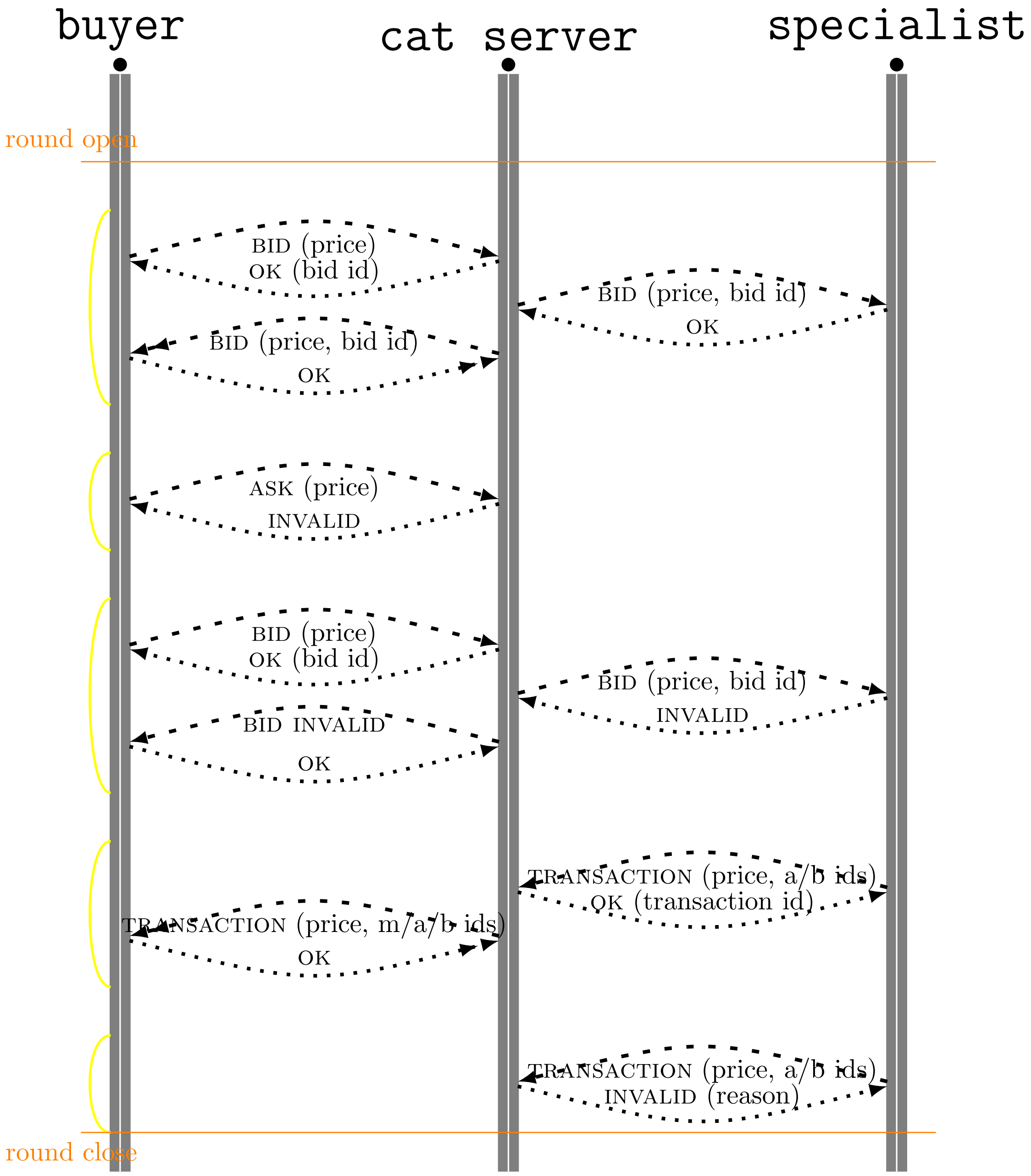

Figure 3: Buyer's dialog during a trading round.

11.4 GET messages

Figure 4 shows GET request sent by CATP clients to the CATP server. The diagram does not show what happens when a client sends a GET request before the game starts. Before the game begins, the server can either return an INVALID response or just an empty list with a text stating that the game has not yet began. The GET request can occur in any order and the clients (traders and specialists) are not aware of each other's GET request.

References

[1] Tac market design: Planning and specification.

[2] D. Cliff. Minimal-intelligence agents for bargaining behaviours in market-based environments. Technical Report HP-97-91, Hewlett-Packard Research Laboratories, Bristol, England, 1997.

[3] S. Gjerstad and J. Dickhaut. Price formation in double auctions. *Games and Economic Behaviour*, 22:1–29, 1998.

[4] D. K. Gode and S. Sunder. Allocative efficiency of markets with zero-intelligence traders: Market as a partial sustitute for individual rationality. *The Journal of Political Economy*, 101(1):119–137, February 1993.

[5] Hypertext Transfer Protocol – HTTP/1.1: Notational Conventions and Generic Grammar.

[6] A. E. Roth and I. Erev. Learning in extensive-form games: Experimental data and simple dynamic models in the intermediate term. *Games and Economic Behavior*, 8:164–212, 1995.

[7] J. Rust, J. H. Miller, and R. Palmer. Behaviour of trading automata in a computerized double auction market. In D. Friedman and J. Rust, editors, *The Double Auction Market: Institutions, Theories and Evidence*, Santa Fe Institute Studies in the Sciences of Complexity, chapter 6, pages 155–199. Perseus Publishing, Cambridge, MA, 1993.

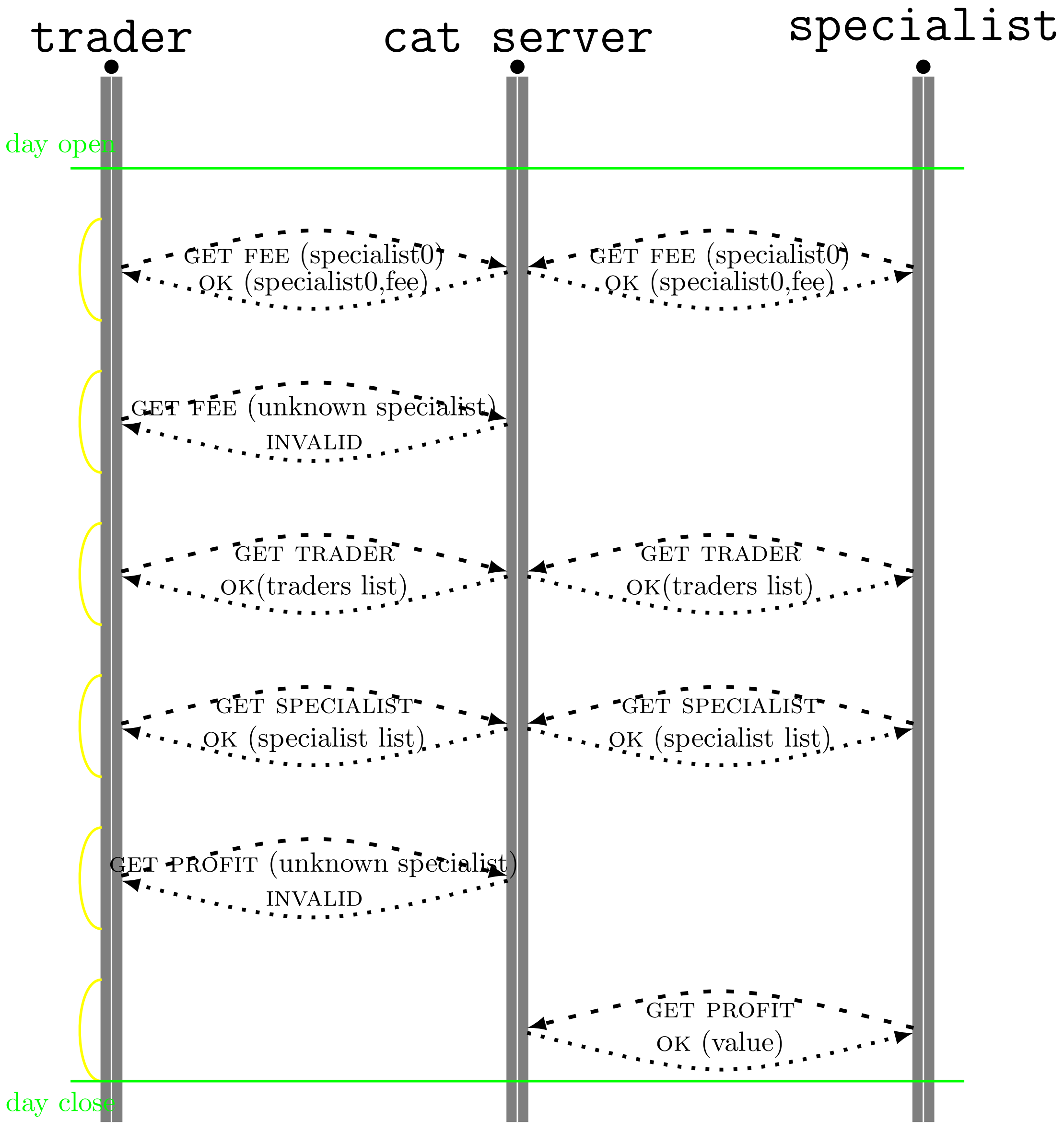

Figure 4: Possible `GET` requests during a day.