Extreme Word Guessing Game

Protocol Definition

Overview

This document defines the communication protocols for an interactive multi-player game, called the *Extreme Word Guessing Game*. The system needs includes two kinds of software components (processes) that will communicate with each other, namely a *Player* and a *Word Server*. The next section outlines the various types of conversations that may occur between these components and the general communication patterns that these conversations follow. It also defines messages that the protocols involves. The section after that defines how the software components must encode and decode them so they understand each other.

Conversations, Communication Patterns, and Messages

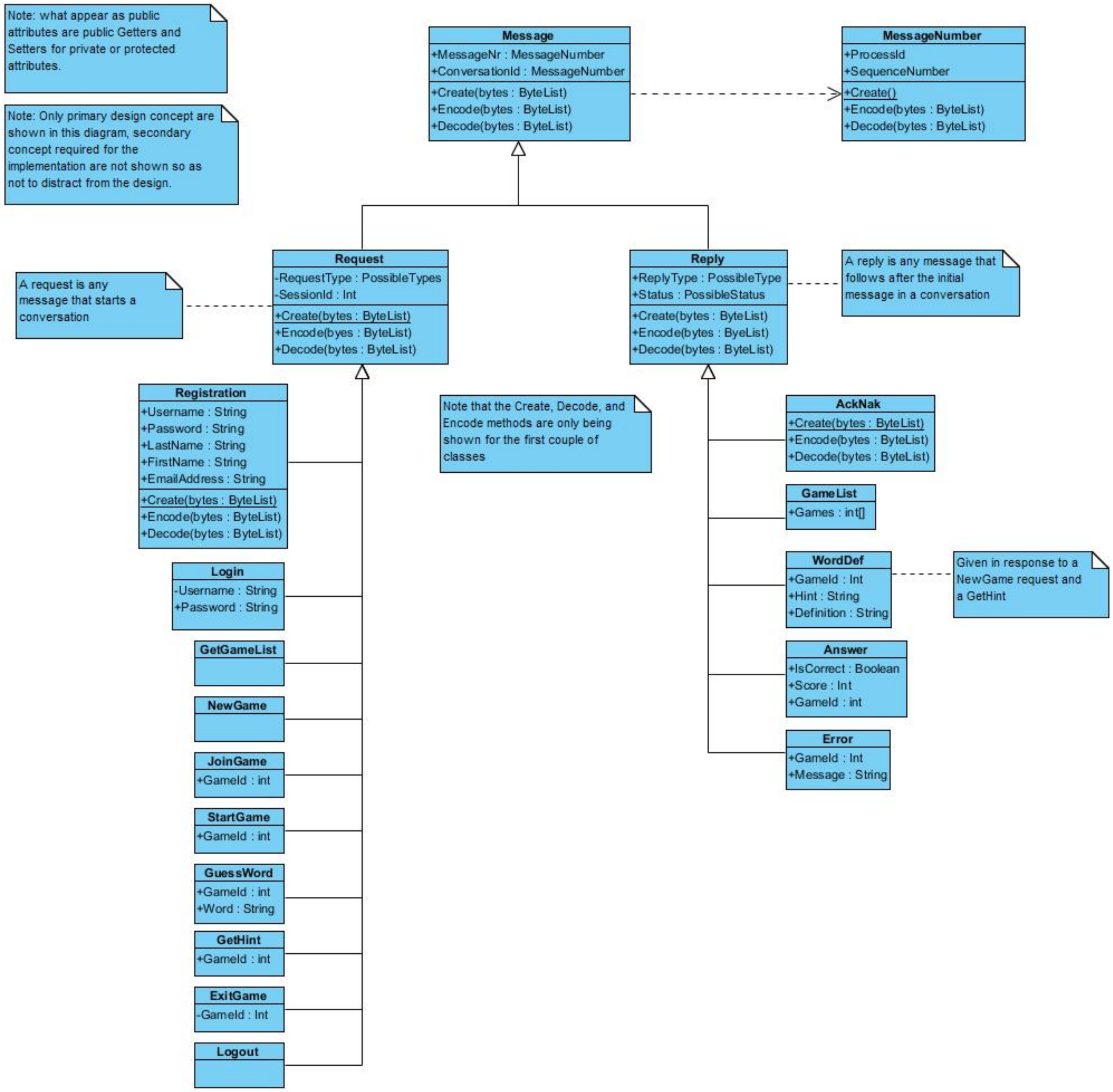
Table 1 lists the possible types of conversations involved in this system, along with which component initiates the conversation, other components involved, and general communication pattern. The communication pattern defines the possible message sequences in both normal and abnormal conditions. (Note that this game is a simple client-server system with only two component and one communication pattern. Nevertheless, this documentation uses a slightly more robust table to illustrate one way of describing a more complex set of protocols.) The only communication pattern used here is *Request-Reply* with various request messages that come from the list of specialization of the *Request* class in Figure 1 and reply messages that come from the list of specializations of the *Reply* class in Figure 1. Figures 2-4 illustrate the possible message sequences for the *Request-Reply* pattern.

Table 1 – Conversations and Protocols for the *Extreme Word Guessing Game*

Protocol / Conversation	Initiator	Other Participants	Communication Pattern
Registration *registers a new user*	Player	Word Server	*Request-Reply*, with *Registration* and *AckNak* as the messages
Login *Logs a user in and creates a session, during which the user can create or join multiple games*	Player	Word Server	*Request-Reply*, with *Login* and *AckNak* as the messages
ListGames	Player	Word Server	*Request-Reply*, with *GetGameList* and *GameList* as the messages
NewGame *Creates a new game*	Player	Word Server	*Request-Reply*, with *NewGame* and *WordDef* as the messages

JoinGame *Joins the player to an existing game, discover from the list of existing games.*	Player	Word Server	*Request-Reply*, with *JoinGame* and *WordDef* as the messages
StartGame *Starts a game that has at least two players and hasn't been started yet.*	Player	Word Server	*Request-Reply*, with *StartGame* and *AckNak* as the messages
GuessWord *Allows the player to guess a word in a game and know if it right or not.*	Player	Word Server	*Request-Reply*, with *GuessWord* and *Answer* as the messages
GetHint *Allows the player to request another hint in a game.*	Player	Word Server	*Request-Reply*, with *GetHint* and *WordDef* as the messages
ExitGame *Exists the player from a game*	Player	Word Server	*Request-Reply*, with *ExitGame* and *AckNak* as the messages
Logout *Log a player out, terminate that player's session*	Player	Word Server	*Request-Reply*, with *Logout* and *AckNak* as the messages

Figure 1 – Message Classes



Note: what appear as public attributes are public Getters and Setters for private or protected attributes.

Note: Only primary design concept are shown in this diagram, secondary concept required for the implementation are not shown so as not to distract from the design.

A request is any message that starts a conversation

A reply is any message that follows after the initial message in a conversation

Note that the Create, Decode, and Encode methods are only being shown for the first couple of classes

Given in response to a NewGame request and a GetHint

Figure 2 – A successful a Request-Reply conversation

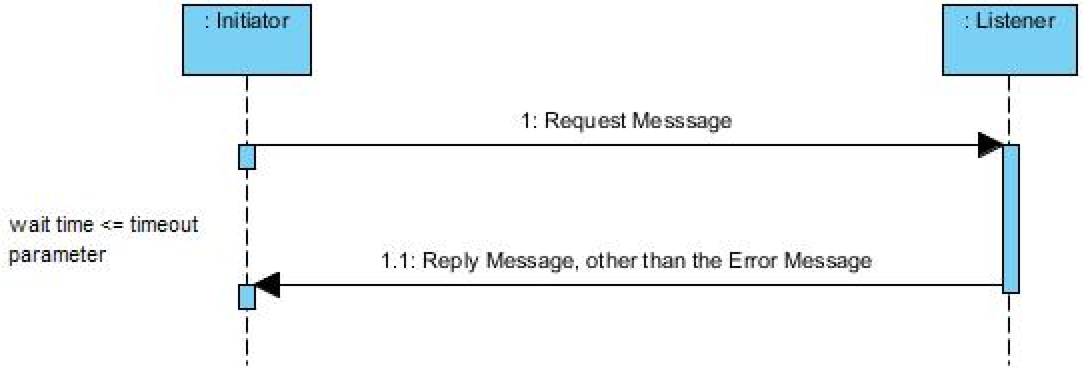

: Initiator
: Listener
1: Request Messsage
wait time <= timeout parameter
1.1: Reply Message, other than the Error Message

Figure 3 – A timeout situation for a Request-Reply conversation

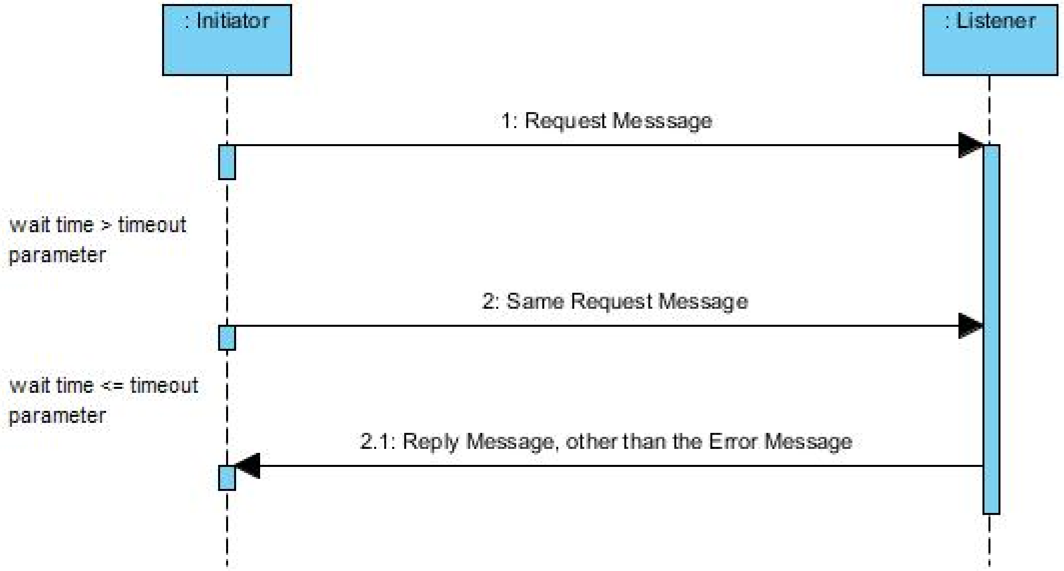
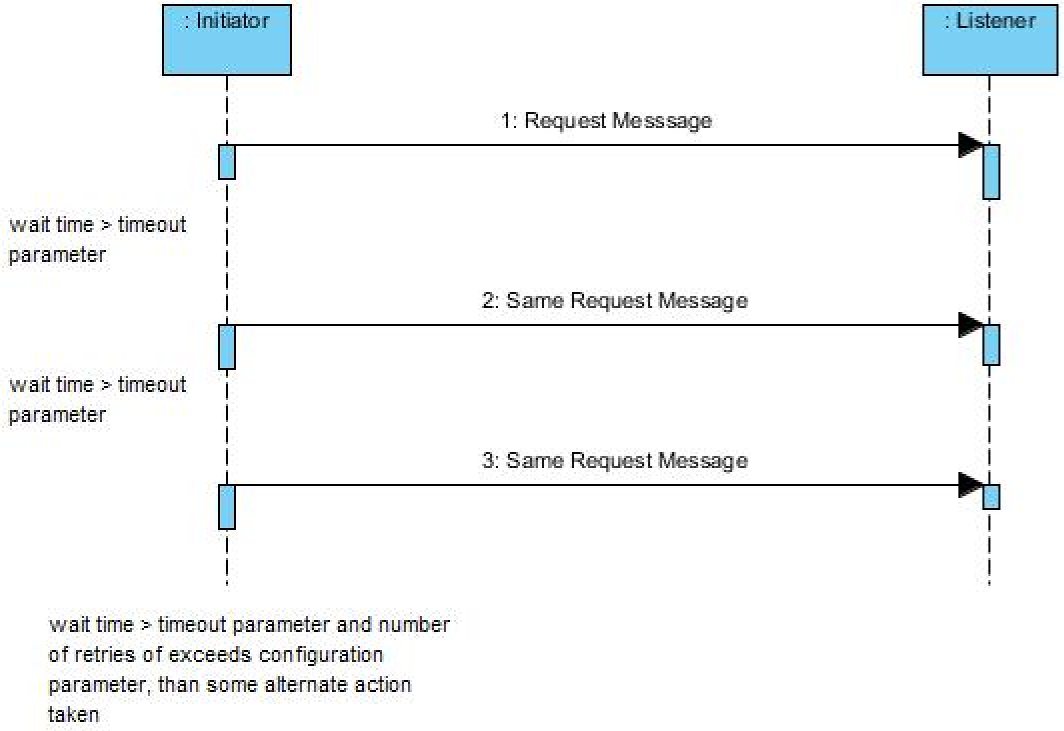

: Initiator
: Listener
1: Request Messsage
wait time > timeout parameter
2: Same Request Message
wait time <= timeout parameter
2.1: Reply Message, other than the Error Message

Figure 4 – An aborted Request-Reply conversation


: Initiator
: Listener
1: Request Messsage
wait time > timeout parameter
2: Same Request Message
wait time > timeout parameter
3: Same Request Message
wait time > timeout parameter and number of retries of exceeds configuration parameter, than some alternate action taken

Message Encoding / Decoding

A message will be encoded recursively using the following rules:

1. The encoding of a *Message* object involves writing its Class Id, the length of its encoded properties, and its properties into a ByteList.
   1.1. The encoding properties process is a pre-defined order of the class
   1.2. Each property is encoded as follows:
      1.2.1. A primitive numeric value (e.g. an integer) is written out in network byte order
         1.2.1.1. Byte – 1 byte
         1.2.1.2. Int16 – 2 bytes
         1.2.1.3. Int32 – 4 bytes
         1.2.1.4. Int64 – 8 bytes
         1.2.1.5. Single Precision Real – 4 bytes
         1.2.1.6. Double Precision Real – 4 bytes
      1.2.2. A char is encoded by writing a two-byte Unique representation of the char value.
      1.2.3. A string is encoded by writing out its length as an Int16 (in network byte order) and a sequence of bytes, where the bytes are a Unicode representation of the string.
      1.2.4. A Boolean value is written out as a byte with a value of 0 (false) or 1 (true)

1.2.5. An array or list of primitive values is encoded by first writing out the count of elements in the array or list as an Int16 (in network byte order), followed by an encoding of each value following rules 1.2.1 – 1.2.4

1.2.6. A property whose value is object is encoded following Rule 1 recursively.

1.2.7. An array or list of objects is encoded by first writing out the count of elements in the array or list as an Int16 (in network byte order), followed by an encoding of each object following Rule 1

Message Semantics

To Be Written