

Protocol Definition

Overview

This document defines the communication protocols for an interactive multi-player game, called the *Brilliant Students vs. Zombie Professors* (or *BSvZP*). The system will consist of a variety of software components (processes) that will communicate with each other, namely: *Game (G)*, *Playing Field (PF)*, *Clock Tower (CT)*, *Brilliant Students (BS)*, *Excuse Generators (EG)*, *Whining Spinners (WS)*, *Zombie Professors (ZP)*, a *Monitor (M)*, and *Referee (R)*. 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.

The different communication patterns the protocols in Table 1 include *One-way Send*, *Request-Reply*, *Unreliable Multicast*, *Reliable Multicast*, *Ongoing Update Stream*, and *WebMethod Innovation*. Each communication patterns, each the last, use the messages that come from the list of specialization of the *Request* and *Reply* class in Figure 1. Figures 2 - 9 illustrate the possible message sequences for the *One-way Send*, *Request-Reply*, and *3-party XYZ* patterns.

Table 1 – Conversations and Protocols for the BSvZP

(Note: protocols in gray will be implemented later.)

Protocol / Conversation	Initiator	Other Participants	Communication Pattern, Messages, and Semantics
Game Registration <i>Registers a new game so others can discover it.</i>	G	GR	<i>WebMethod Invocation</i> of the <i>GR.GameRegister</i> method, which will take the game's label, end point, and status as parameters. This method will

			return a unique game id and a block of id's that can be assigned to agents that join the game.
Game Status Change	G	GR	<i>WebMethod Invocation</i> of the <i>GR.GameStatusChange</i> method, which will take the game's registry id and the new status, and optionally the name of the winner if the status is "Completed".
Get Games	BS, EG, or WS	GR	<i>WebMethod Invocation</i> of the <i>GR.GetGames</i> method, which will take a status as a parameter and return an array of all of the games (<i>GameInfo</i> objects) with that status.
Join Game	BS, EG, WS, ZP, or R	G	<i>Request-Reply</i> , with <i>JoinGame</i> and <i>AckNak</i> as request and reply messages, where <ul style="list-style-type: none"> The <i>AgentInfo</i> attribute of the <i>JoinGame</i> request contains a <i>ComponentInfo</i> object and that object with only the <i>AgentType</i> specified If the <i>Status</i> of the <i>AckNak</i> message is <i>Success</i>, then the <i>ObjResult</i> in the <i>AckNak</i> message will be a completed <i>ComponentInfo</i> object. If the <i>Status</i> of the <i>AckNak</i> message is <i>Failure</i>, the agent could not join the game for some reason and the <i>Message</i> of the <i>AckNak</i> contains the specific reason or error message.
Remove Component	R	G	<i>Request-Reply</i> , with <i>RemoveComponent</i> and <i>AckNak</i> messages <ul style="list-style-type: none"> The <i>ComponentId</i> attribute of the <i>RemoveComponent</i> is the identifier of the component to remove from the playing field. If the <i>Status</i> of the <i>AckNak</i> message is <i>Success</i>, then the <i>IntResult</i> of the <i>AckNak</i> is the component's Id. Otherwise, the request failed and the <i>Message</i> of the <i>AckNak</i> contains the specific reason or error message.
Start Game	G	BS, EG, WS, ZP, or R	Reliable Multicast, with <i>StartGame</i> as the initial message, a <i>Ready</i> as agent reply, and <i>Proceed</i> the last message.
End Game	R G	G BS, EG, WS, or ZP	Unreliable Multicast, with <i>EndGame</i> as the message.
Get Configuration	BS, EG, WS, ZP, M, or R	G	<i>Request-Reply</i> , with <i>GetResource</i> and <i>ConfigurationReply</i> messages, where <ul style="list-style-type: none"> The <i>GetType</i> in the <i>GetResource</i> message is <i>Game Configuration</i>. If the <i>Status</i> of the <i>ConfigurationReply</i> message is <i>Success</i>, then the <i>Config</i> object of the <i>ConfigurationReply</i> is a <i>Configuration</i> object.

			<ul style="list-style-type: none"> Otherwise, the request failed and the <i>Config</i> is null.
Get Playing Field Layout	<i>BS, EG, WS, ZP, or R</i>	<i>G</i>	<p><i>Request-Reply</i>, with <i>GetResource</i> and <i>PlayingFieldLayout</i> messages, where</p> <ul style="list-style-type: none"> The <i>GetType</i> in the <i>GetResource</i> message is <i>Playing Field Layout</i>. If the <i>Status</i> of the <i>PlayingFieldLayout</i> message is <i>Success</i>, then the resource of the <i>PlayingFieldLayout</i> is a <i>PlayingFieldLayout</i> object. Otherwise, the request failed and the <i>PlayingFieldLayout</i> is null.
Get Brilliant Student List	<i>BS, EG, WS, ZP, or R</i>	<i>G</i>	<p><i>Request-Reply</i>, with <i>GetResource</i> and <i>AgentListReply</i> messages, where</p> <ul style="list-style-type: none"> The <i>GetType</i> in the <i>GetResource</i> message is <i>Brilliant Student List</i>. If the <i>Status</i> of the <i>AgentListReply</i> message is <i>Success</i>, then the <i>AgentList</i> of the <i>AgentListReply</i> is a <i>ComponentList</i> object containing <i>ComponentInfo</i> objects about all <i>BrilliantStudent</i> objects currently on the playing field. Otherwise, the request failed and the <i>AgentList</i> is null.
Get Excuse Generator List	<i>BS, EG, WS, ZP, or R</i>	<i>G</i>	<p><i>Request-Reply</i>, with <i>GetResource</i> and <i>AgentListReply</i> messages, where</p> <ul style="list-style-type: none"> The <i>GetType</i> in the <i>GetResource</i> message is <i>Excuse Generator List</i>. If the <i>Status</i> of the <i>AgentListReply</i> message is <i>Success</i>, then the <i>AgentList</i> of the <i>AgentListReply</i> is a <i>ComponentList</i> object containing <i>ComponentInfo</i> objects about all <i>ExcuseGenerator</i> objects currently on the playing field. Otherwise, the request failed and the <i>AgentList</i> is null.
Get Whining Spinner List	<i>BS, EG, WS, ZP, or R</i>	<i>G</i>	<p><i>Request-Reply</i>, with <i>GetResource</i> and <i>AgentListReply</i> messages, where</p> <ul style="list-style-type: none"> The <i>GetType</i> in the <i>GetResource</i> message is <i>Excuse Generator List</i>. If the <i>Status</i> of the <i>AgentListReply</i> message is <i>Success</i>, then the <i>AgentList</i> of the <i>AgentListReply</i> is a <i>ComponentList</i> object containing <i>ComponentInfo</i> objects about all <i>WhiningSpinner</i> objects currently on the playing field. Otherwise, the request failed and the <i>AgentList</i> is null.
Get Zombie Professor List	<i>BS, EG, WS, ZP, or R</i>	<i>G</i>	<p><i>Request-Reply</i>, with <i>GetResource</i> and <i>AgentListReply</i> messages, where</p> <ul style="list-style-type: none"> The <i>GetType</i> in the <i>GetResource</i> message is <i>Excuse Generator List</i>. If the <i>Status</i> of the <i>AgentListReply</i> message is <i>Success</i>, then the <i>AgentList</i> of the <i>AgentListReply</i> is a <i>ComponentList</i> object containing <i>ComponentInfo</i> objects about all <i>ZombieProfessor</i> objects currently on the playing field. Otherwise, the request failed and the <i>AgentList</i> is null.

Get Excuse	BS	EG	<i>Request-Reply</i> , with <i>GetResource</i> and <i>ResourceReply</i> messages, where <ul style="list-style-type: none"> The <i>GetType</i> in the <i>GetResource</i> message is <i>Excuse</i>. If the <i>Status</i> of the <i>ResourceReply</i> message is <i>Success</i>, then the <i>Resource</i> of the <i>ResourceReply</i> is an <i>Excuse</i> object. Otherwise, the request failed and the <i>Resource</i> is null.
Get Whining Twine	BS	EG	<i>Request-Reply</i> , with <i>GetResource</i> and <i>ResourceReply</i> messages, where <ul style="list-style-type: none"> The <i>GetType</i> in the <i>GetResource</i> message is <i>Whining Twine</i>. If the <i>Status</i> of the <i>ResourceReply</i> message is <i>Success</i>, then the <i>Resource</i> of the <i>ResourceReply</i> is a <i>Whining Twine</i> object. Otherwise, the request failed and the <i>Resource</i> is null.
Send Out Time Tick	CT	BS, EG, WS, or ZP	<i>Unreliable Multicast</i> , with <i>TickMessage</i> as the messages.
Validate Tick	PF	CT	<i>Request-Reply</i> , with <i>ValidateTick</i> and <i>AckNak</i> messages, where <ul style="list-style-type: none"> The <i>ComponentId</i> attribute in <i>ValidateTick</i> message is the identify of the component that wants to use the <i>Tick</i> If the <i>Status</i> of the <i>AckNak</i> message is <i>Success</i>, then the tick is valid. Otherwise, the request failed and the <i>Message</i> of the <i>AckNak</i> contains the specific reason or error message.
Move	BS or ZP	G, CT	<i>Request-Reply</i> , with <i>Move</i> and <i>AckNak</i> messages, where <ul style="list-style-type: none"> The <i>ComponentId</i> attribute in the <i>Move</i> message is the identify of the component that wants to use the <i>Tick</i> The <i>ToSquare</i> attribute in the <i>Move</i> message is where the agent (BS or ZP) wants to move The <i>EnablingTick</i> attribute in the <i>Move</i> message is a valid <i>Tick</i> that agent hasn't used for any other purpose. If the <i>Status</i> of the <i>AckNak</i> message is <i>Success</i>, then the move took place. Otherwise, the request failed and the <i>Message</i> of the <i>AckNak</i> contains the specific reason or error message.
Throw Bomb	BS	G	<i>Request-Reply</i> , with <i>Throw Bomb</i> and <i>AckNak</i> messages, where <ul style="list-style-type: none"> The <i>ComponentId</i> attribute in the <i>Throw Bomb</i> message is the identify of the component that wants to throw the bomb. The <i>Bomb</i> attribute in the <i>Throw Bomb</i> message has to be bomb containing at least one <i>Excuse</i> and one <i>Whining Twine</i> The <i>TowardsSquare</i> attribute in the <i>Throw Bomb</i> message

			<p>represent the target of the bomb. If the bomb doesn't have enough <i>Whining Twine</i> to go that distance, it will fail short, in some other square.</p> <ul style="list-style-type: none"> • The <i>EnablingTick</i> attribute in the <i>Move</i> message is a valid Tick that agent hasn't used for any other purpose. • If the <i>Status</i> of the <i>AckNak</i> message is <i>Success</i>, then the bomb was thrown (but possibly not all the way to the target. The <i>ObjResult</i> attribute contains a <i>Square</i> object that describes where the bomb landed. • Otherwise, the request failed and the <i>Message</i> of the <i>AckNak</i> contains the specific reason or error message.
Eat	ZP	G	<p><i>Request-Reply</i>, with <i>Eat</i> and <i>AckNak</i> messages, where</p> <ul style="list-style-type: none"> • The <i>ZombiId</i> attribute in the <i>Eat</i> message is the identity of the zombie that wants to eat something else. • The <i>TargetId</i> attribute is the identity of the target agent that the zombie wants to eat. • If the <i>Status</i> of the <i>AckNak</i> message is <i>Success</i>, then the <i>Eating</i> took place. • Otherwise, the request failed and the <i>Message</i> of the <i>AckNak</i> contains the specific reason or error message.
Change Strength	G	BS, EG, WS, or ZP	<p><i>Request-Reply</i>, with <i>ChangeStrength</i> and <i>AckNak</i> messages, where</p> <ul style="list-style-type: none"> • The <i>DeltaValue</i> attribute is the delta value that needs to be apply to the receiving agent's current strength. • If the <i>Status</i> of the <i>AckNak</i> message is <i>Success</i>, then the operation was successful. • Otherwise, the request failed and the <i>Message</i> of the <i>AckNak</i> contains the specific reason or error message.
Collaborate	BS	BS	<p><i>Request-Reply</i>, with <i>Collaborate</i> and <i>AckNak</i> messages, where</p> <ul style="list-style-type: none"> • If the <i>Status</i> of the <i>AckNak</i> message is <i>Success</i>, then the <i>ObjResult</i> attribute contains <i>ComponentInfo</i> object that describes the current target of the receiving agent. • Otherwise, the request failed and the <i>Message</i> of the <i>AckNak</i> contains the specific reason or error message.
GetStatus	M or R	BS, EG, WP, or ZB	<p><i>Request-Reply</i>, with <i>GetStatus</i> and <i>StatusReply</i> messages, where</p> <ul style="list-style-type: none"> • If the <i>Status</i> of the <i>StatusReply</i> message is <i>Success</i>, then the

			<p>StatusInfo attribute contains ComponentInfo object that describes the current status of the receiving agent.</p> <ul style="list-style-type: none"> Otherwise, the ComponentInfo Message of the StatusReply is null.
Agent Update Stream			<p>Ongoing Update Stream. With StartUpdateStream, AckNak, AgentListReply, and EndUpdateStream as message.</p> <ul style="list-style-type: none"> An agent will send the game a StartUpdateStream message to start the update stream. The game will send back an AckNak message with the Status if the stream is ready to start. After that, the game will send AgentListReply messages periodically, until the agent sends a StopUpdateStream message.

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 – 8 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 first represented from a byte containing a “1” for True or a “0” for False. A true means that the object is present and its encoding follows. A false means the object is not present. The encoding of the objects follows 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

Figure 01 - Message Classes for Word Guessing

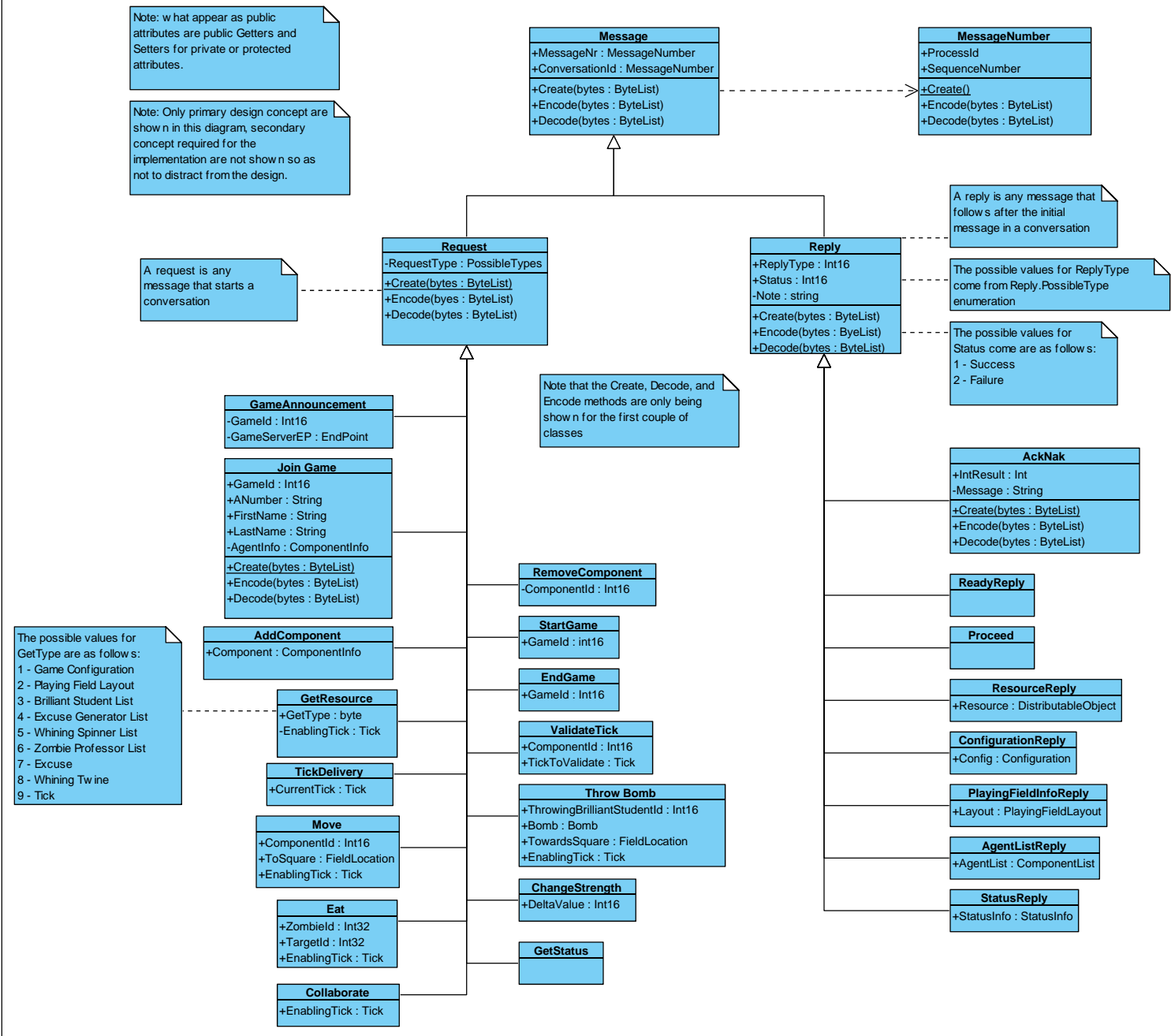
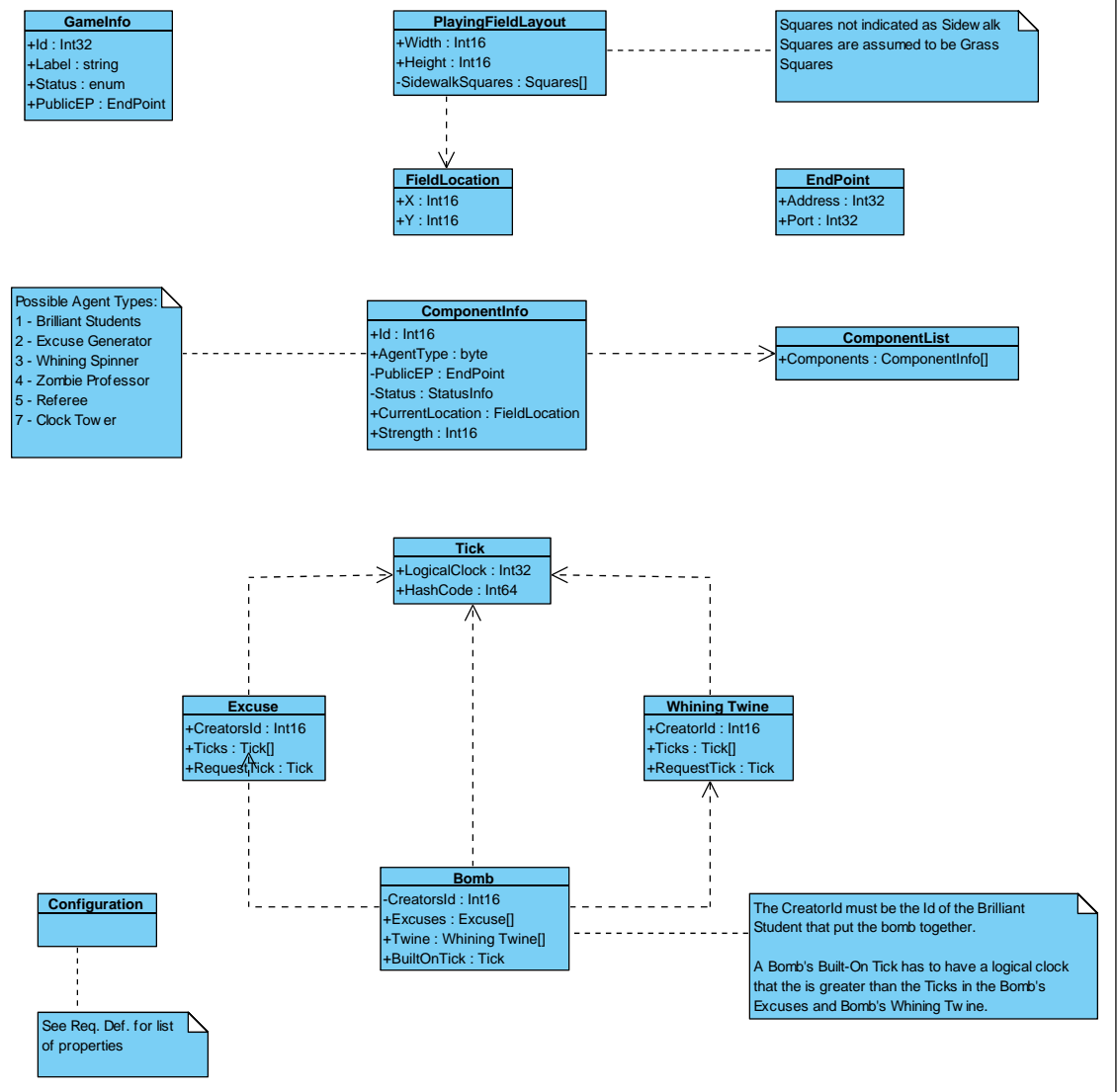
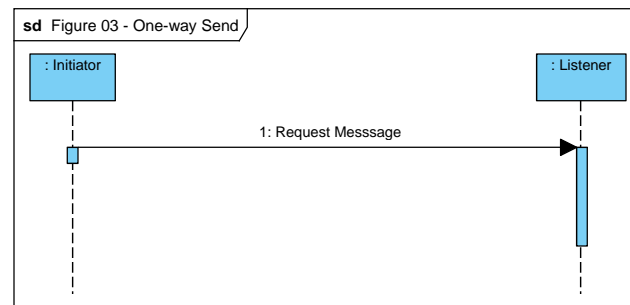
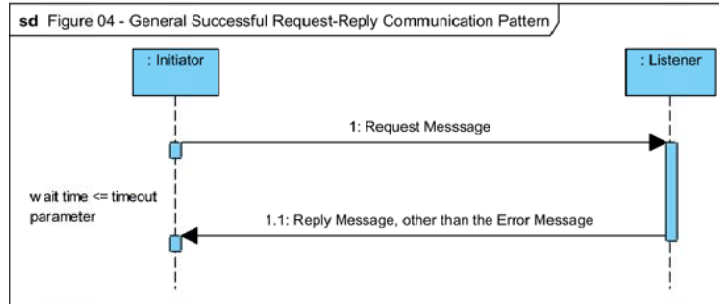
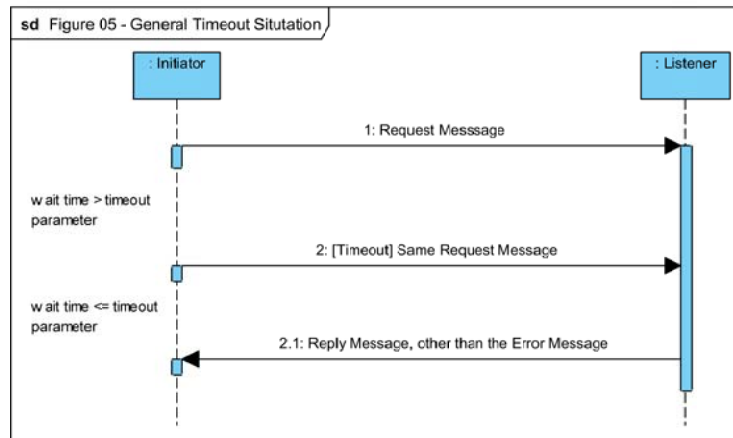


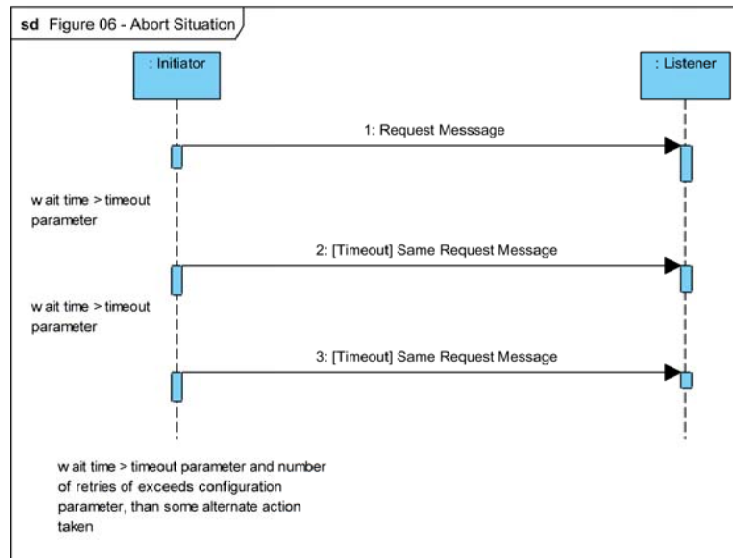
Figure 02 - Support Classes

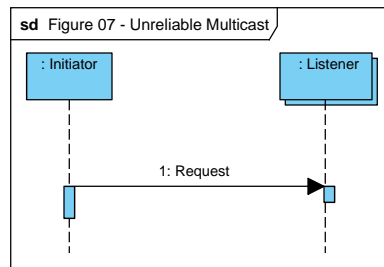


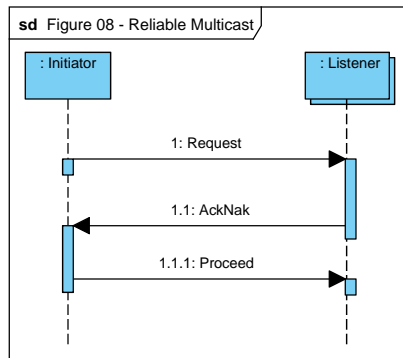












sd Figure 09 - Ongoing Update Stream

