Stranger Danger (SD) – Communication Protocols and Architecture Design

By Shem Hale, Sarah Isaac, and Katie Sweet

1. Overview

SD includes 5 different processes: Client Process (CliP), Main Servers (MS), Statistics Server (SS), Registry (R), and Camera Process(CamP). This document defines how these processes communicate with each other. Table 1 provides a summary list of all the protocols within the system and the remaining sections define these more in depth.

Table 1 – Protocol List

Purpose	Initiator	Other Processes	Pattern
Register	CliP, MS, SS, CamP	R	RR
Request Statistics	CliP	MS, SS	Frontend + ISM + Ack
Raw Data Query	CliP	MS, SS	ISM + Ack
Sync Data	MS	MS	RR
Get Main Server List	CliP,CamP,SS	R,MS	Proxy + Multicast
Transfer Motion Image	CamP	MS	RR
Alive	R	MS	RR

Note: All patterns will communicate with UDP

2. Messages and Shared Objects

Figures 1-4 show a class hierarchy of all the messages used in SD protocols and Tables 2-5 provides some additional details about each message class's attributes and their meaning.

Figure 1 - Abstract Message Classes

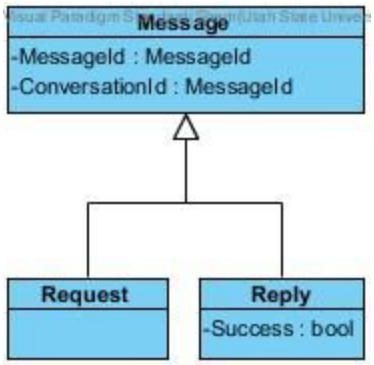


Figure 2 - Request Message Classes

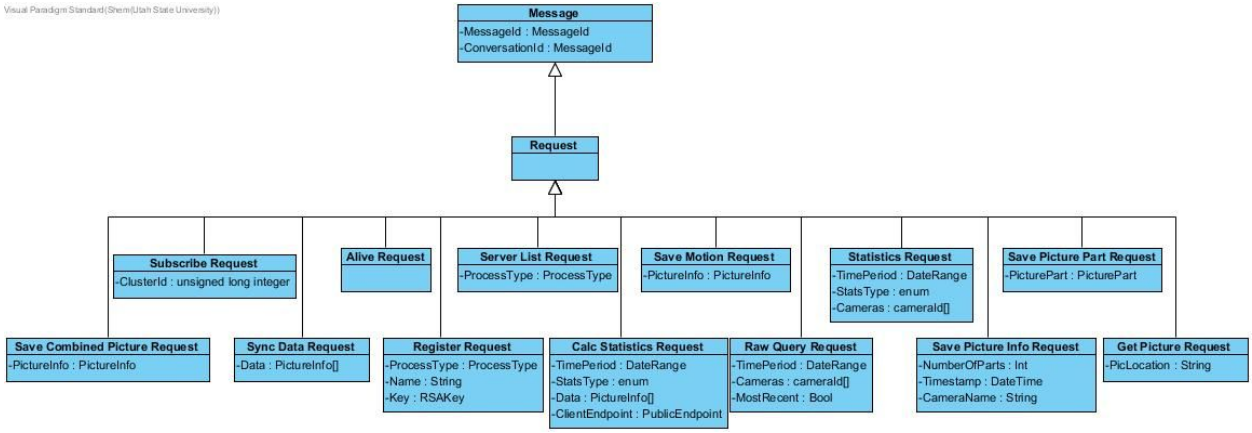


Figure 3 - Reply Message Classes

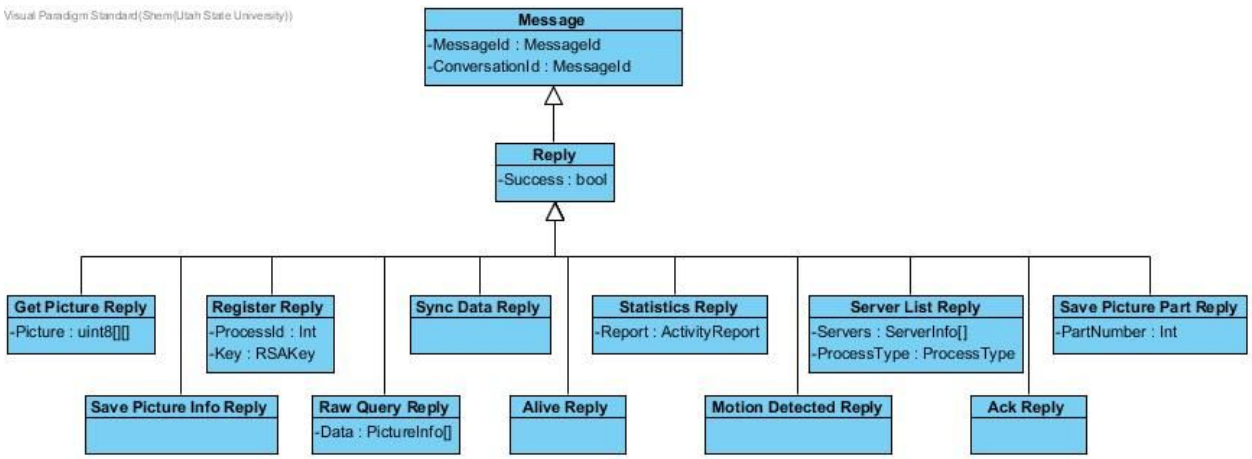


Figure 4 - Shared Objects

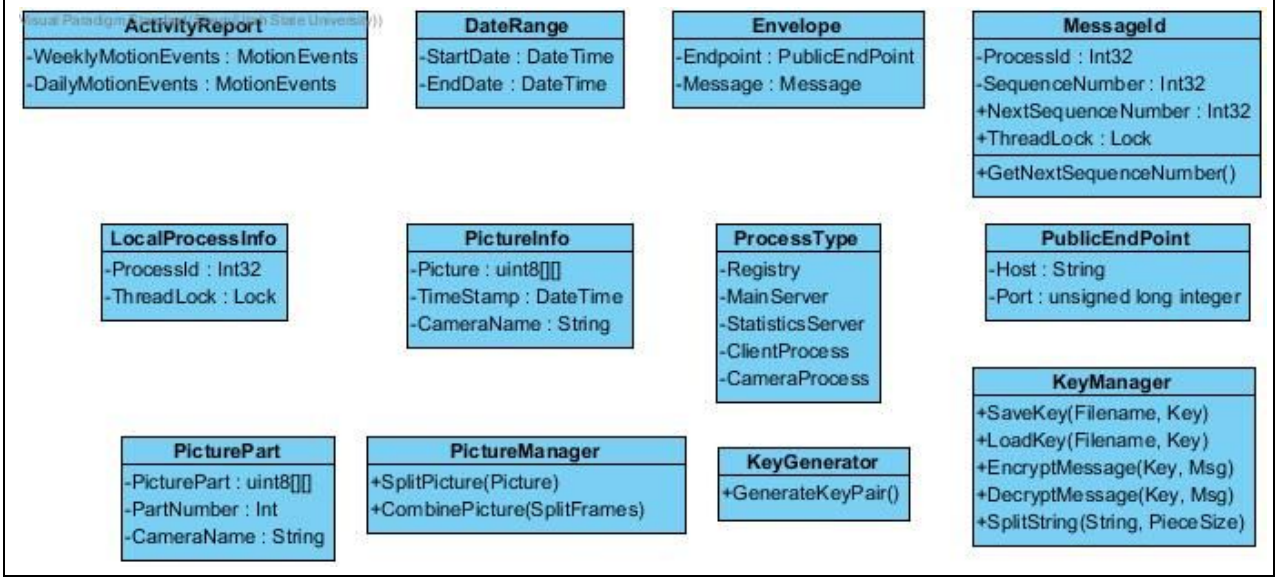


Table 2 – Request Message Class Descriptions

Note: All concrete request message classes inherit from Message and Request, and therefore include the MessageId and ConversationId attributes. Some message classes contain no other attributes.

Class Name	Attribute	Attribute Type	Meaning
(all request messages)	MessageId	MessageId	A unique identifier for a message.
	ConversationId	MessageId	A unique identifier for a conversation. The conversation's id is the same as the message number for the first message in the conversation.
SyncDataRequest:#Messages	Data	PictureInfo	An object that holds the resource that should be synced.
AliveRequest:#Messages	--	--	--
RegisterRequest:#Messages	ProcessType	ProcessType	A type that identifies the type of process.
	Name	String	An identifier for a Camera Process.
	Key	RSAKey	An RSA key that holds a process' public key for the Registry to use.
ServerListRequest:#Messages	ProcessType	ProcessType	A type that identifies what type of processes the list should contain.
SaveMotionRequest:#Messages	PictureInfo	PictureInfo	An object that holds the information about a picture such as the timestamp, camera's id that took the picture, and the camera's cluster id.
RawQueryRequest:#Messages	TimePeriod	DateRange	An object that holds a start and end date.
	Cameras	CameraId[]	An array of cameras.
	MostRecent	Bool	A boolean indicating whether the Client would like only the most recent picture or not.
StatisticsRequest:#Messages	TimePeriod	DateRange	An object that holds a start and end date.

	Cameras	CameraId[]	An array of cameras.
	StatsType	Enum	A type that defines the type of report that should be generated.
CalcStatisticsRequest:#Messages	TimePeriod	DateRange	An object that holds a start and end date.
	Cameras	CameraId[]	An array of cameras.
	Data	PictureInfo[]	An array containing picture data used for a statistics report.
	ClientProcess	Endpoint	An object containing the endpoint for a Client process so the Statistics server knows where to send the message to.
SaveCombinedPictureRequest:#Messages	PictureInfo	PictureInfo	An object that holds the reassembled picture and information about the picture, such as the timestamp and the associated camera.
SavePictureInfoRequest:#Messages	NumberOfParts	Int	The number of parts that the picture was split into.
	Timestamp	DateTime	The date and time that the picture was taken.
	CameraName	String	The name of the camera that took the picture.
SavePicturePartRequest:#Messages	PicturePart	PicturePart	An object that contains a part of a picture and information about the part, such as the position of the part and the associated camera's name.
GetPictureRequest:#Messages	PicLocation	String	A string indicating where a picture is located in the database.

Table 3 – Reply Message Class Descriptions

Note: All concrete request message classes inherit from Message and Reply, and therefore include the MessageId, ConversationId, and Success attributes. Some message classes contain no other attributes.

Class Name	Attribute	Attribute Type	Meaning
(all reply messages)	MessageId	MessageNumber	A unique identifier for a message.
	ConversationId	MessageNumber	A unique identifier for a conversation. The conversation's id is the same as the message number for the first message in the conversation.
	Success	Bool	True if the requested action was successfully completed; otherwise false.
Reply:#Message	--	--	--
RegisterReply:#Messages	ProcessId	ProcessId	A unique id used to identify a process for communication.
	Key	RSAKey	A public RSA key that belongs to the process that requested to be registered.
RawQueryReply:#Messages	Data	PictureInfo[]	An array containing raw picture data.
SyncDataReply:#Messages	--	--	--
AliveReply:#Messages	--	--	--
StatisticsReply:#Messages	Report	ActivityReport	An object that contains a statistics report on picture data
MotionDetectedReply:#Messages	--	--	--
ServerListReply:#Messages	Servers	ServerInfo[]	An array containing information about Servers
	ProcessType	ProcessType	The type of process that the server list contains.
AckReply:#Messages	--	--	--
SavePictureInfoReply:#Messages	--	--	--
SavePicturePartReply:#Messages	PartNumber	Int	The part number of a picture that was received.
GetPictureReply:#Messages	Picture	uint8[][]	An object that contains a picture.

Table 3 – Shared Object Class Descriptions

Instances of these classes are parts of messages and therefore their serialization will be embedded in message serializations.

Class Name	Attribute	Attribute Type	Meaning
MessageId	ProcessId	Uint32	A unique identifier for the process that created this message number.
	SequenceNumber	Uint32	A number from a circular sequence that is unique within the context of the process. If MessageId B is created immediately following MessageId A on the same process, then A.SequenceNumber + 1 = B.SequenceNumber, until the number equals Uint32.Max. At this point, B.SequenceNumber will be set to 1.
PublicEndPoint	Host	String	A representation of the host's IP Address.
	Port	Int	The end point's port number.
PictureInfo	Picture	Uint8[][]	A 2d array containing information about a picture.
	TimeStamp	DateTime	An object containing the date and time when the picture was taken.
	CameraName	String	A unique identifier for a camera
DateRange	StartDate	DateTime	The requested start time
	EndDate	DateTime	The requested end time
LocalProcessInfo	ProcessId	Uint8	A unique identifier for a process
Envelope	Endpoint	PublicEndPoint	The endpoint of either where the message should be sent to or where the message came from
	Message	Message	The message that is to be exchanged
PicturePart	PicturePart	Uint8[][]	A part of a split picture.
	PartNumber	Int	The order of the picture part.
	CameraName	String	The name of the camera that took the picture.

3. Communication Patterns

The communication patterns that Stranger Danger uses are Request-Reply, Proxy, Proxy + Software Multicast, Intermediate State Message(ISM) + Acknowledge(Ack), and Frontend + ISM + Ack. These are described here and referenced in the Communication Protocols Section.

3.1 Request-Reply

Figure 5 – Request and Reply Communication Pattern

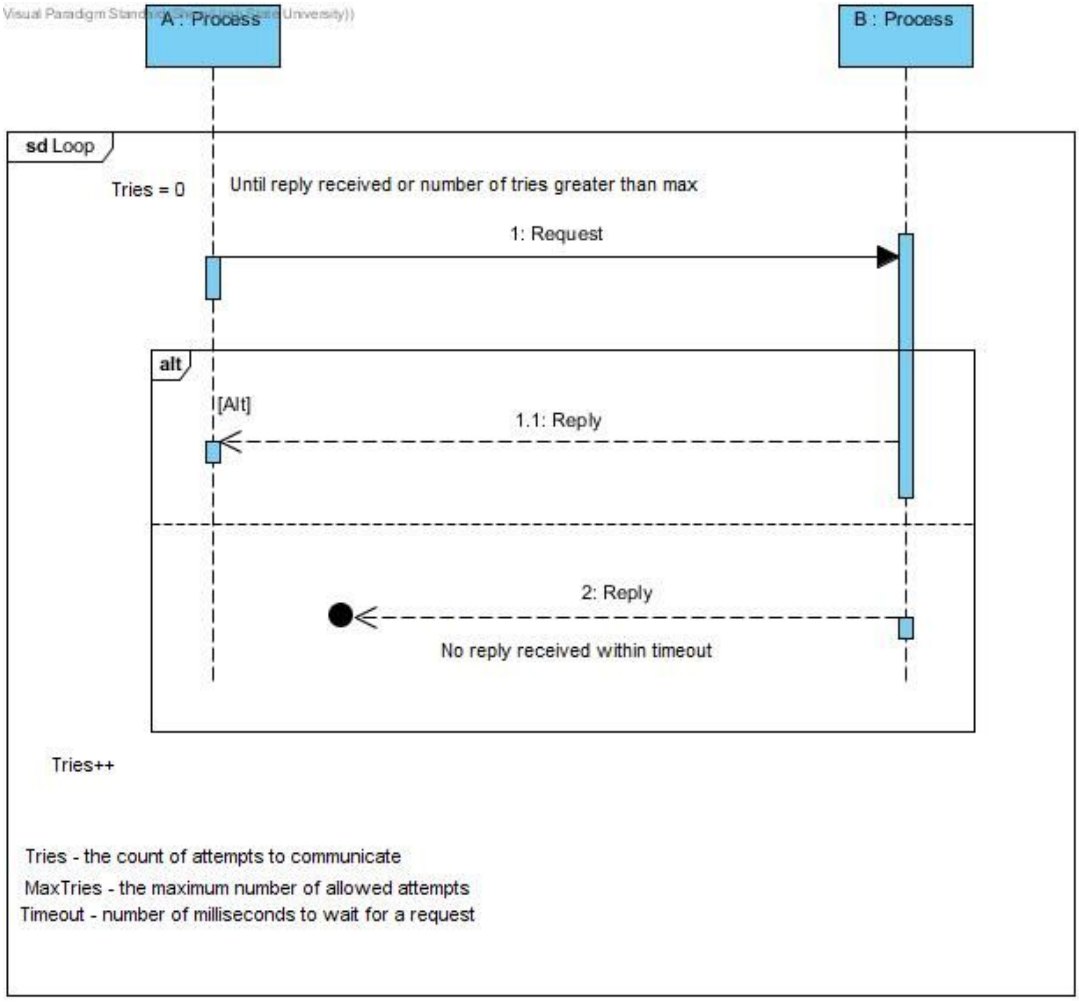


Figure 5 shows a basic sequence diagram for a request-reply pattern with the following substitutable concepts:

- An initiating process (A)
- A receiving process (B)
- A message (Request) that A sends to B to start the conversation
- A message (Reply) that B returns after receiving the request
- A MaxTries parameter that limits the number of attempts that will be made before the conversation is aborted if not successfully completed with a reply.
- Timeout parameter that specifies how long the initiator will wait for a reply.

3.2 Proxy

Figure 6 – Proxy Communication Pattern

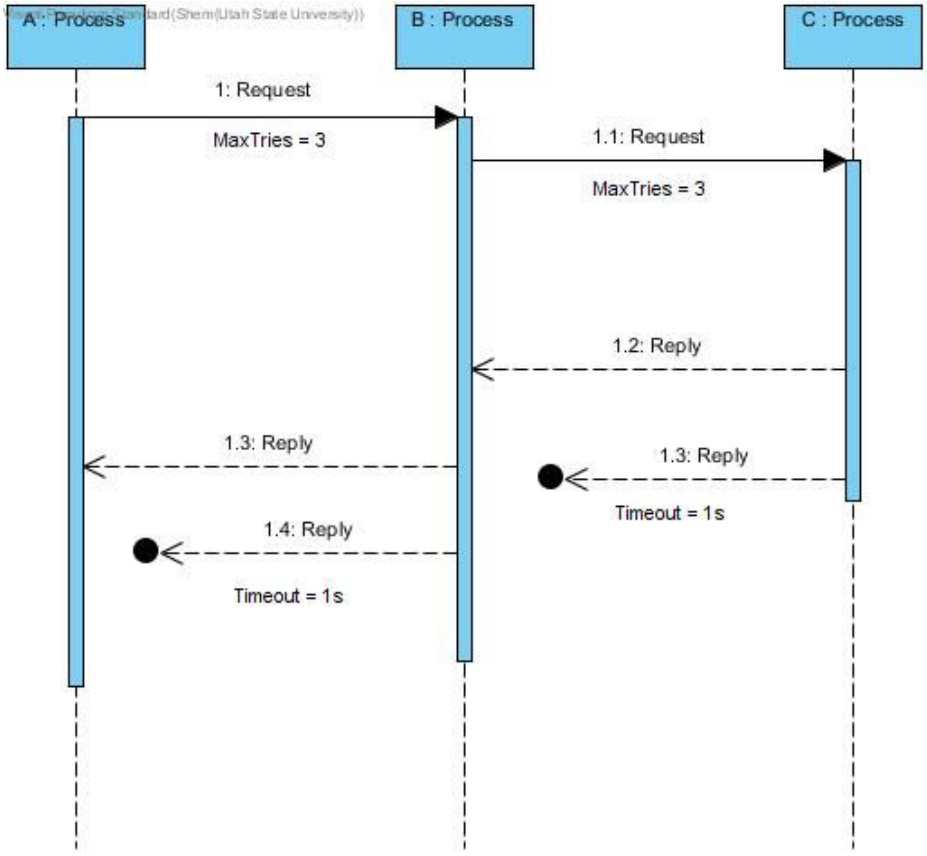


Figure 6 shows a basic sequence diagram for a proxy pattern with the following substitutable concepts:

- An initiating process (A)
- An intermediate receiving process (B)
- A receiving process (C)
- A message (Request) that A sends to B to start the conversation
- A message (Request) that B sends to C
- A message (Reply) that C returns after receiving the request from B
- A message (Reply) that B returns after receiving the reply from C
- A MaxTries parameter that limits the number of attempts that will be made before the conversation is aborted if not successfully completed with a reply.
- Timeout parameter that specifies how long the initiator will wait for a reply.

3.3 Proxy + Software Multicast

Figure 7 – Proxy and Software Multicast Communication Pattern

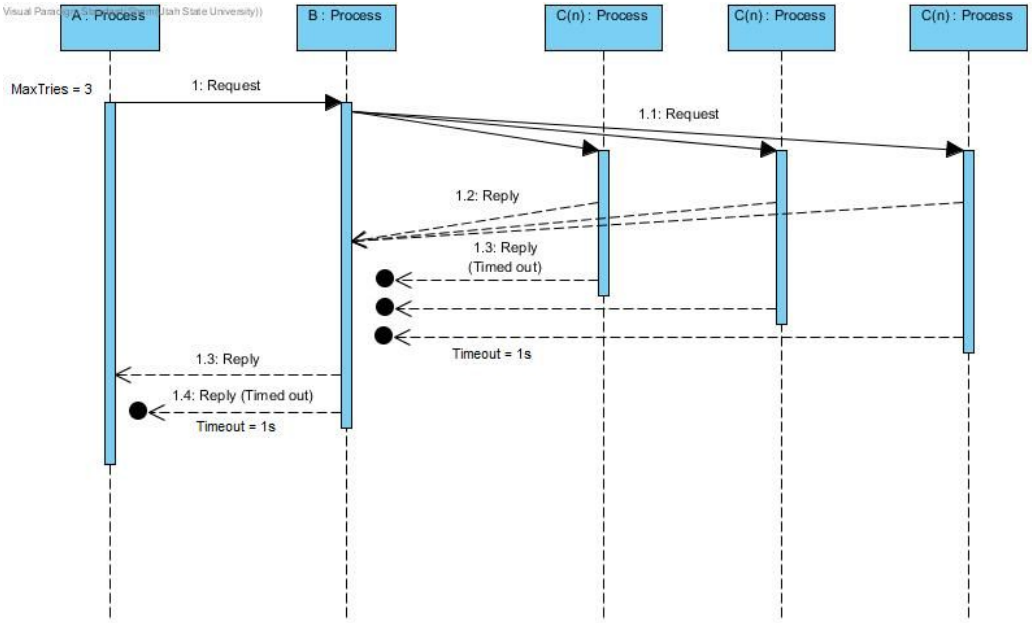


Figure 7 shows a basic sequence diagram for a proxy and software multicast pattern with the following substitutable concepts:

- An initiating process (A)
- An intermediate receiving process (B)
- Multiple receiving processes (C(n))
- A message (Request) that A sends to B to start the conversation
- A message (Request) that B multicasts to C(n)
- A message (Reply) that C(n) returns after receiving the request from B
- A message (Reply) that B returns after receiving the reply from C
- A MaxTries parameter that limits the number of attempts that will be made before the conversation is aborted if not successfully completed with a reply.
- Timeout parameter that specifies how long the initiator will wait for a reply.

3.4 Intermediate State Message + Acknowledge

Figure 8 – Intermediate State Message and Acknowledge Communication Pattern

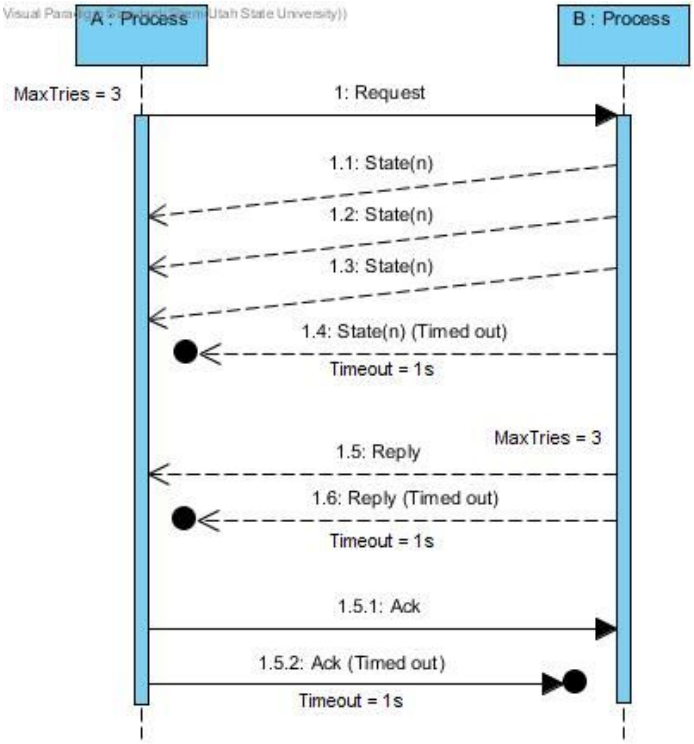


Figure 8 shows a basic sequence diagram for an intermediate state message and acknowledge pattern with the following substitutable concepts:

- An initiating process (A)
- A receiving process (B)
- A message (Request) that A sends to B to start the conversation
- A message (State(n)) that B returns after receiving the request from A, containing the progress of a task that is being run on B
- A message (Reply) that B returns after the task is done
- A MaxTries parameter that limits the number of attempts that will be made before the conversation is aborted if not successfully completed with a reply.
- Timeout parameter that specifies how long the initiator will wait for a reply.

3.5 Frontend + Intermediate State Message + Acknowledge

Figure 9 – Frontend and Intermediate State Message and Acknowledge Communication Pattern

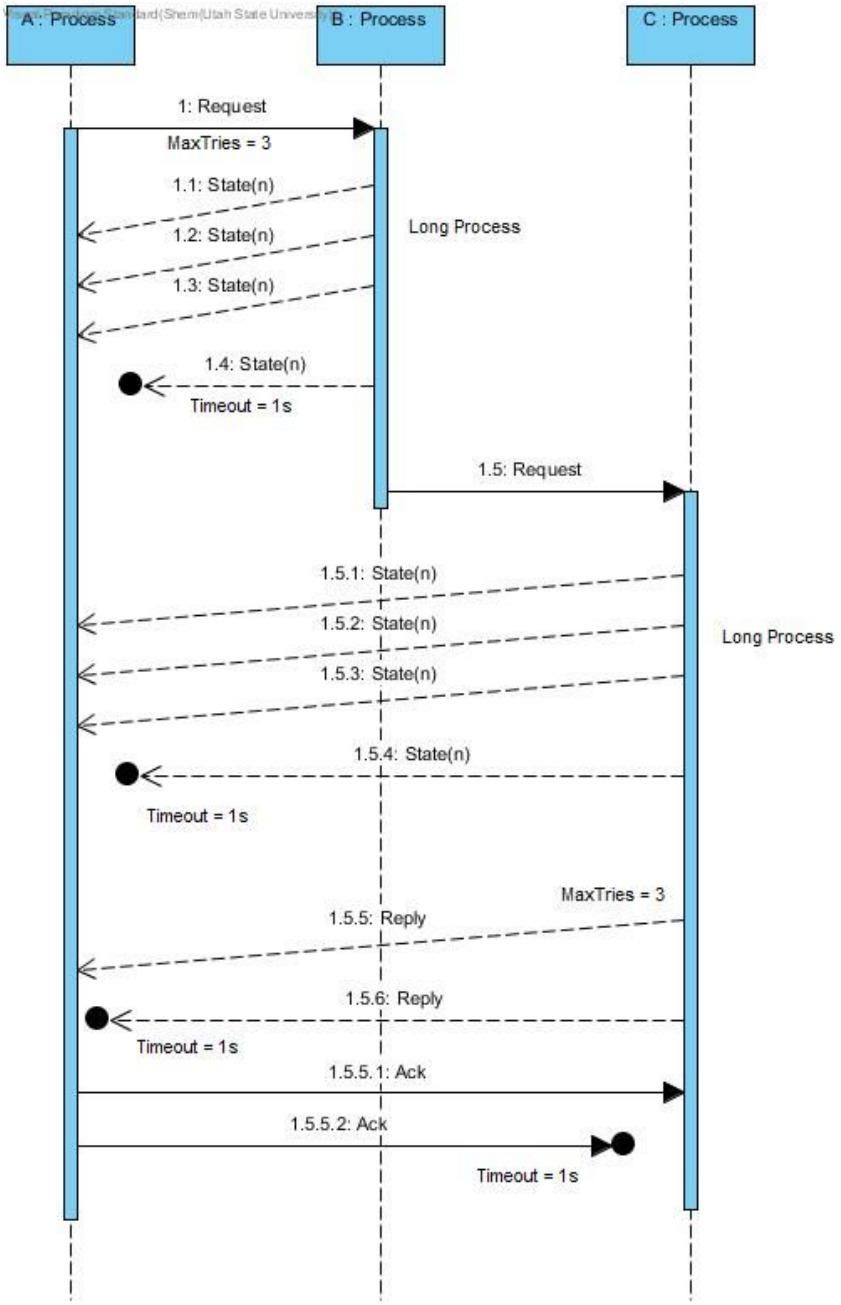


Figure 9 shows a basic sequence diagram for a frontend and intermediate state message and acknowledge pattern with the following substitutable concepts:

- An initiating process (A)
- An intermediate receiving process (B)
- A receiving process (C)
- A message (Request) that A sends to B to start the conversation
- A message (State(n)) that B returns after receiving the request from A, containing the progress of a task that is being run on B
- A message (Request) that B sends to C after the task is done
- A message (State(n)) that C returns after receiving the request from B, containing the progress of a task that is being run on C
- A message (Reply) that C sends to A after the task is done
- A message (Ack) that A returns to C after receiving the reply
- A MaxTries parameter that limits the number of attempts that will be made before the conversation is aborted if not successfully completed with a reply.
- Timeout parameter that specifies how long the initiator will wait for a reply.

4. Communication Protocols

Below are the detailed descriptions of the conversations outlined in Table 1.

4.1 Register (Request-Reply)

Used by a Client Process to register an account.

Message Sequence:

Client Process -> (Register Request) -> Registry

Registry -> (Register Reply) -> Client Process

Semantics and Behaviors:

- A Client initiates the conversation by sending a RegisterRequest message to the Registry. The Registry will then get the next available ProcessId.
- The Registry will send a RegisterReply message with the available ProcessId to the Client indicating that the request was successful.

4.2 Request Statistics (Frontend+ISM+Ack)

Used by a Client Process to get statistics on motion activity for a certain date range

Message Sequence:

Client Process -> (Statistics Request) -> Main Server

Main Server -> (State) -> Client Process

Main Server -> (Calc Statistics Request) -> Statistics Server

Statistics Server -> (State) -> Client Process

Statistics Server -> (Statistics Reply) -> Client Process

Semantics and Behaviors:

- A Client initiates the conversation by sending a StatisticsRequest message to a Main Server. The Client must specify a date range, the type of statistic to be generated, and the group of cameras that the Client is interested in.
- After a Client sends the StatisticsRequest, a Main Server will query the database to retrieve the requested info while sending a State message to let the Client know that the conversation is still alive.
- Once the info is retrieved, the Main Server will send the Statistics Server a CalcStatisticsRequest message. The Statistics Server will get the endpoint of the Client Process from the Main Server and send a State message to the Client while generating the report.
- Once the report is generated, the Statistics Server will send a StatisticsReply message to the Client Process.

4.3 Raw Data Query (ISM+Ack)

Used by a Client Process to get raw data on motion activity for a certain date range.

Message Sequence:

Client Process -> (Raw Query Request) -> Main Server

Main Server -> (State) -> Client Process

Main Server -> (Raw Query Reply) -> Statistics Server

Semantics and Behaviors:

- A Client initiates the conversation by sending a RawQueryRequest message to a Main Server. The Client must specify a date range, the type of statistic to be generated, and the group of cameras that the Client is interested in.
- After a Client sends the StatisticsRequest, a Main Server will query the database to retrieve the requested info while sending a State message to let the Client know that the conversation is still alive.
- Once the info is retrieved, the Main Server will send the Client Process a RawQueryReply message.

4.4 Sync Data (Request-Reply)

Used by a Main Server to sync data with another Main Server.

Message Sequence:

Main Server -> (Sync Data Request) -> Main Server

Main Server -> (Sync Data Reply) -> Main Server

Semantics and Behaviors:

- A Main Server initiates the conversation by sending a SyncDataRequest message to another Main Server.
- The other Main Server will reply with a SyncDataReply message indicating whether the data was successfully synced or not.
- This protocol will be on a timer that will run every 10 minutes.

4.5 Get Main Server List (Proxy+Multicast)

Used by any Process except for the Registry to get the list of available Main Servers.

Message Sequence:

Process -> (Server List Request) -> Registry

Registry -> (Alive Request) -> Main Servers

Main Servers -> (Alive Reply) -> Registry

Registry -> (Server List Reply) -> Process

Semantics and Behaviors:

- A Process initiates the conversation by sending a ServerListRequest message to the Registry.
- The Registry will then send a AliveRequest message to all of the Main Servers. If a Main Server replys back with a AliveReply message, the Registry will append the Main Server's info to a list.
- After the Registry has received a response from each Main Server or the response timed out, the Registry will send the Process a ServerListReply message containing the information about all of the Main Servers that responded.

4.6 Transfer Motion Image (Request-Reply)

Used by a Camera Process to transfer Pictures from a camera when motion is detected to a database.

Message Sequence:

Camera Process -> (Save Motion Request) -> Intercepted by Comm Sub-System

Picture is split up into pieces

Camera Process -> (Save Picture Info Request) -> Main Server

Main Server -> (Save Picture Info Reply) -> Camera Process

for each piece:

Camera Process -> (Save Picture Part Request) -> Main Server

Main Server -> (Save Picture Part Reply) -> Camera Process

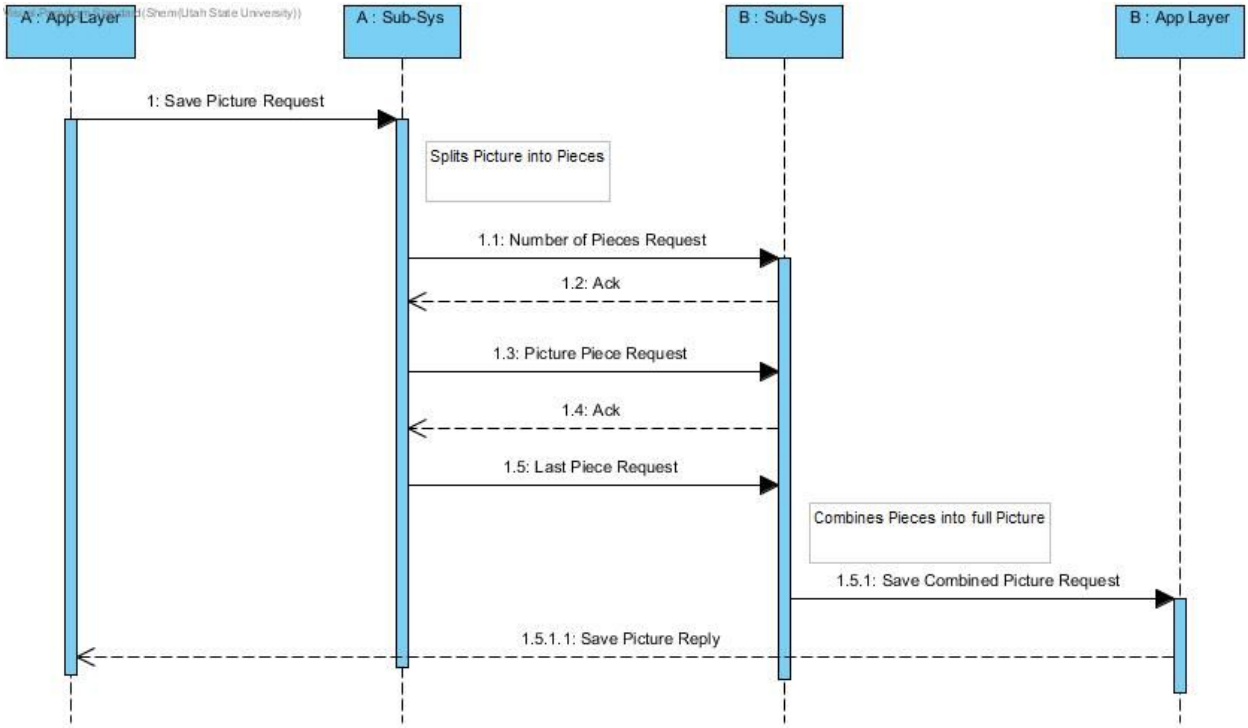
pieces are combined

Main Server -> (Save Combined Picture Request) -> Main Server App Layer

Main Server -> (Motion Detected Reply) -> Camera Process

Semantics and Behaviors:

- A Camera Process initiates the conversation by sending a SaveMotionRequest message. The Communication Sub-System will intercept the message and break up the picture into smaller pieces.
- The Camera Process will then send a SavePictureInfoRequest to the Main Server that lets the server know how many parts will be sent. The Main Server will acknowledge with a SavePictureInfoReply.
- For each piece, the Camera Process will send a SavePicturePartRequest to the Main Server. The Main Server will acknowledge with a SavePicturePartReply when the piece is received.
- Once the last piece is received, the Main Server will combine the pieces and Send a SaveCombinedPictureRequest to the Main Server's App Layer.
- The Main Server will then store the given info in its database and reply with a MotionDetectedReply message indicating whether or not it was successfully stored in the database.



4.7 Alive (Request-Reply)

Used by the Registry to see if any of the Main Servers are still available.

Message Sequence:

Registry -> (AliveRequest) -> Main Server

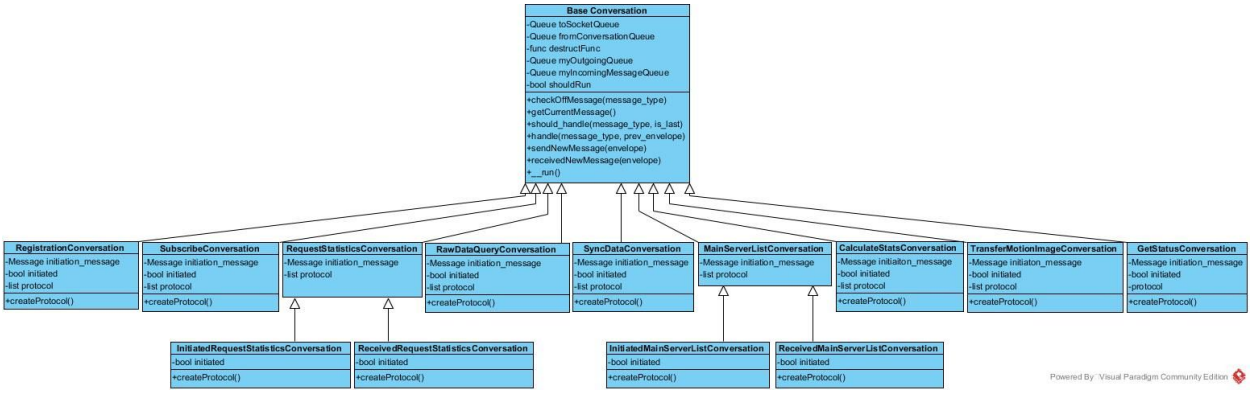
Main Server -> (AliveReply) -> Registry

Semantics and Behaviors:

- The Registry initiates the conversation by sending a AliveRequest message to a Main Server.
- Upon receiving the request, the Main Server will then acknowledge with a AliveReply message to the Registry.
- The Registry will then mark the Main Server as alive if the reply is received. If not, then the Main Server is marked as not alive and potentially removed from the Main Server list.

5. Conversations

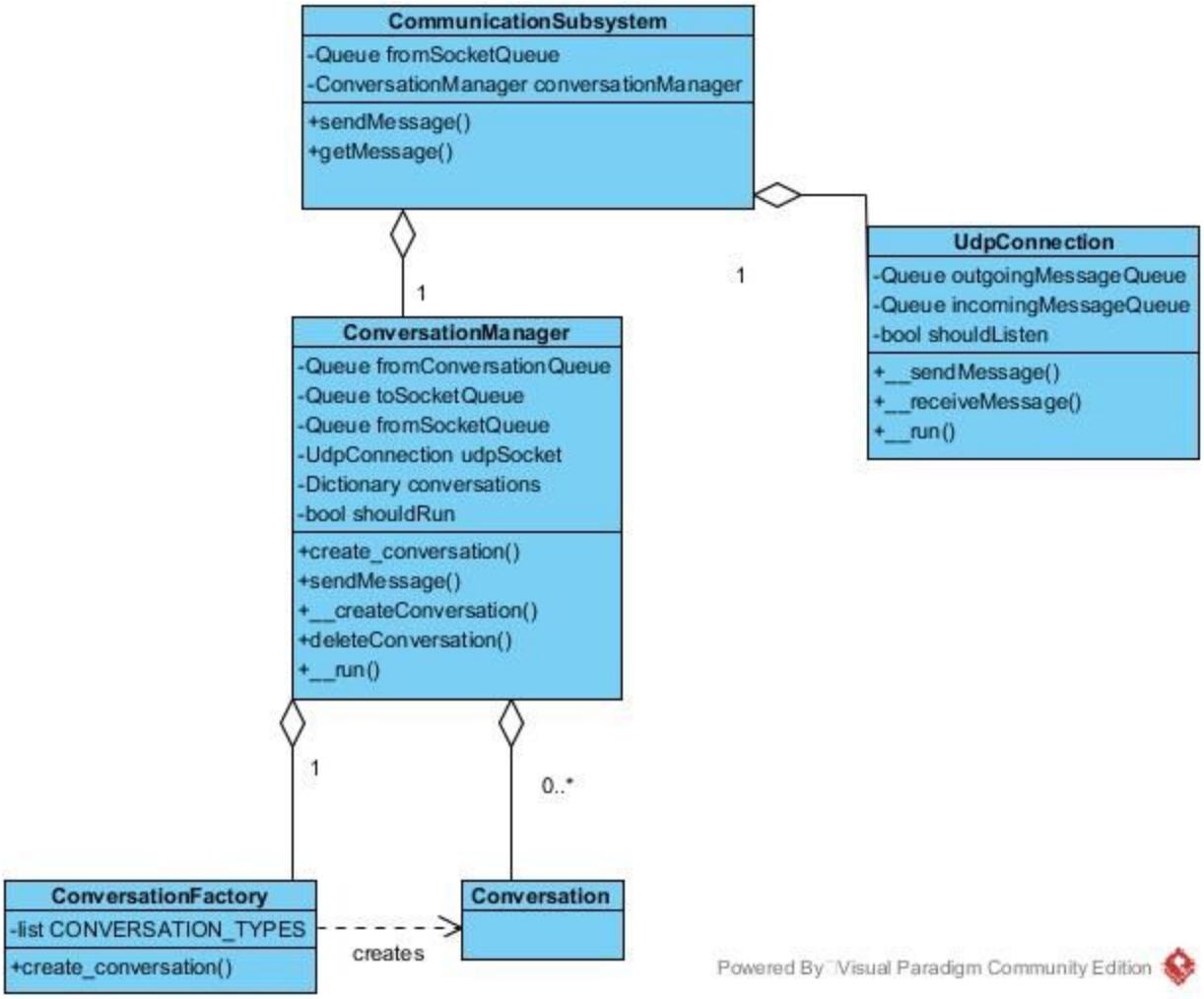
Below is the hierarchy for Conversations



6. Communication Subsystem

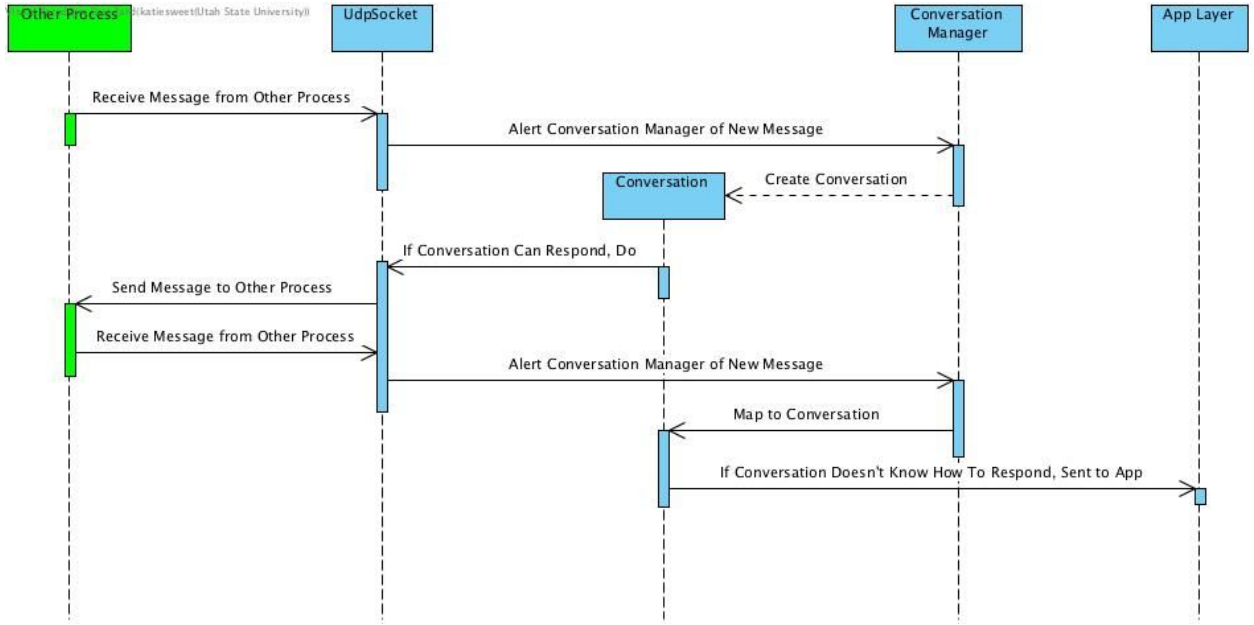
Below are the hierarchy and protocols for the Communication Subsystem

1. **Hierarchy**

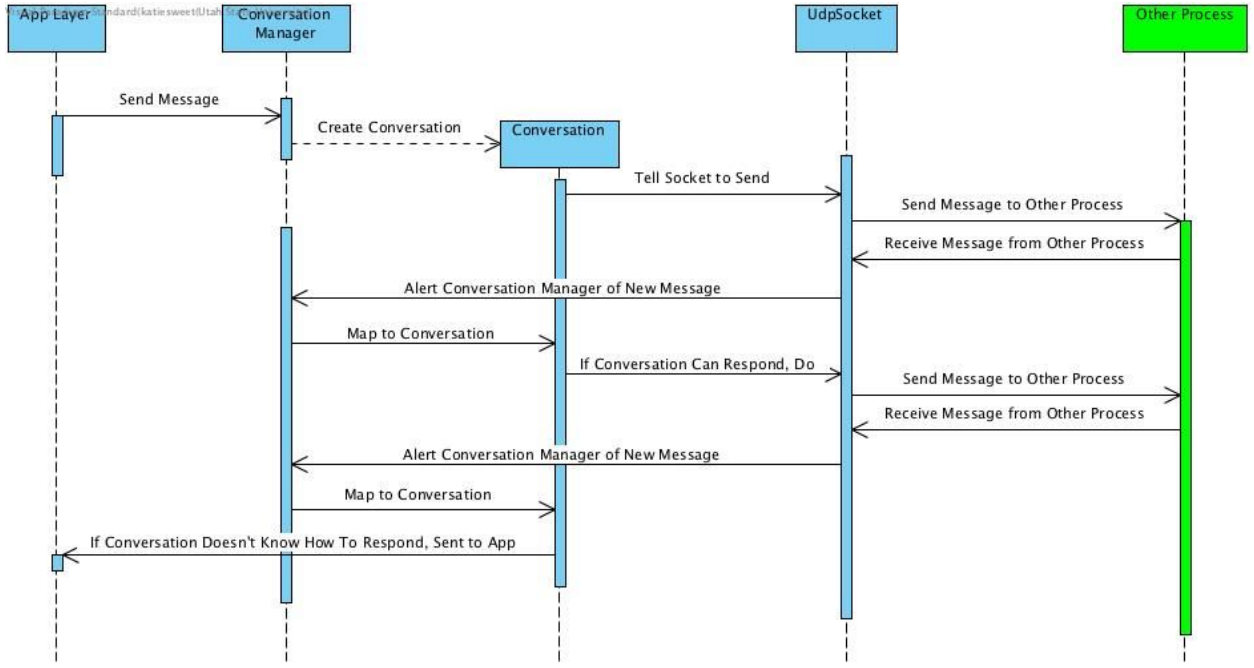


This describes the hierarchy of Conversations and the relation of conversations and communication subsystem.

2. Incoming Message Protocol



3. Outgoing Message Protocol



Each Conversation handles reliability in many ways. If a message is timed out after 1 second, the message will be resent. The max number of resends is 3 for all sent messages. If a message is duplicated or a late message is received, the conversation will assume that the duplicated or late message was never received, so the response to the duplicated or late message will be sent again.

7. Security

The Register protocol has security implemented using RSA public/private keys. Here is the process of encrypting and decrypting the data:

- Every process has a public and private key generated upon Registering. Every process holds the Registry's public key and does not hold the Registry's private key.
- When sending a Register Request, the process encrypts the message with the Registry's public key. The Register Request also contains the Process' public key for the Registry to use.
- After receiving the request, the Registry decrypts the message with the Registry's own private key.
- The Registry then creates a Register Reply message and encrypts the message with the Process' public key that was passed in the message itself.
- After receiving the reply, the Process decrypts the message with the Process' own private key.

Note that each message is too large to encrypt all at once, so each message is split into smaller strings. Each string is then encrypted and combined before being sent. Similar process for decryption.