Controll Developement

Erik Jonsson Thorén

March 27, 2013

Chapter 1

Introduction

This document specifies the goals for the Controll project. It also includes some diagrams over different models and message routing specifications.

`http://github.com/thecopy/Controll`

Chapter 2

Milestones

This document is just a weak specification.

2.1 Controll 0.0.1

The 0.0.1 version of Controll will be a prototype of the fundamental communications between a client and a zombie.

- Simple Authentication.
- Register Zombie
- Register User
- Fetch Zombie Information
  - Owned zombies
- Ping zombie (through server)
- Message delivery verification

2.2 Controll 0.0.2

Changes from 0.0.1:

- Client should be able to
  - Get Installed Activities on specified zombie
  - Activate simple activity[1].
- Zombie should be able to
  - Detect a plugin file (see section 3.1 for a specification of a plugin file)
  - Synchronize the plugin collection with the server

[1] A terminating response from the activity. No further dialog between client and the activity.

  - Execute a plugin

- The server will keep a list of installed plugins for every zombie
- Message queue, MQ, offline support[2]
- More tests for the client and zombie clients.

See section 3.3 for a specification of the message routing in 0.0.2 for a clear overview of how the offline queue support is specified to be implemented.

[2] A message to a zombie or a client will wait until the zombie or client becomes online and then deliver the message

Chapter 3

Appendix

3.1 Specification of a plugin file

A plugin file is a .NET library (.dll) with classes implementing the `Controll¬ .Common.IControllPlugin` interface which declares:

- `void Execute(Controll.Common.IPluginContext)`
- `Guid Key { get; }`
- `string Name { get; }`
- `string CreatorName { get; }`
- `DateTime LastUpdated { get; }`
- `string Description { get; }`

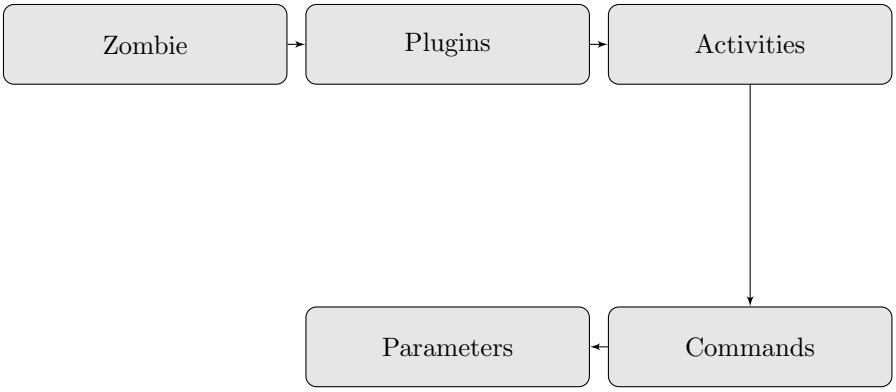
Where `Controll.Common.IPluginContext` is an object containing information about the execution parameters. It declares:

- `IDictionary<string, string> Parameters { get; }`
- `void Started();`
- `void Finish(string result);`
- `void Error(string errorMessage);`
- `void Notify(string message);`

The methods `Error(string)` and `Notify(string)` are functionality planned to be implemented in $\geq$ 0.0.3. It's purpose is to notify the client of an on-going process in the activity (or of course an error). `Started()` is called when the activity is started and `Finish(string)` is called when the activity is completed with the string parameter containing the result(s).

3.2 The Zombie Model

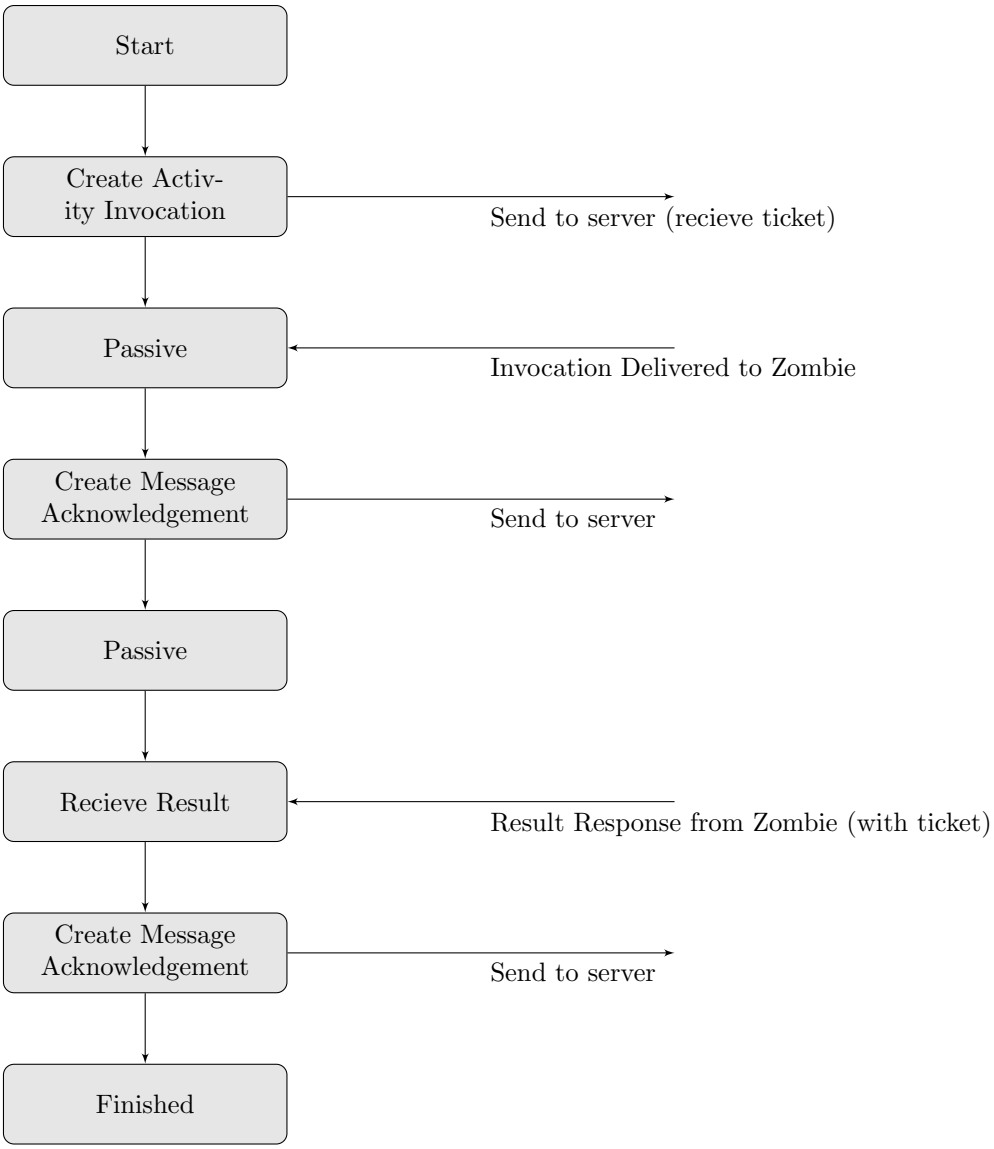
The arrow represents *have many.*



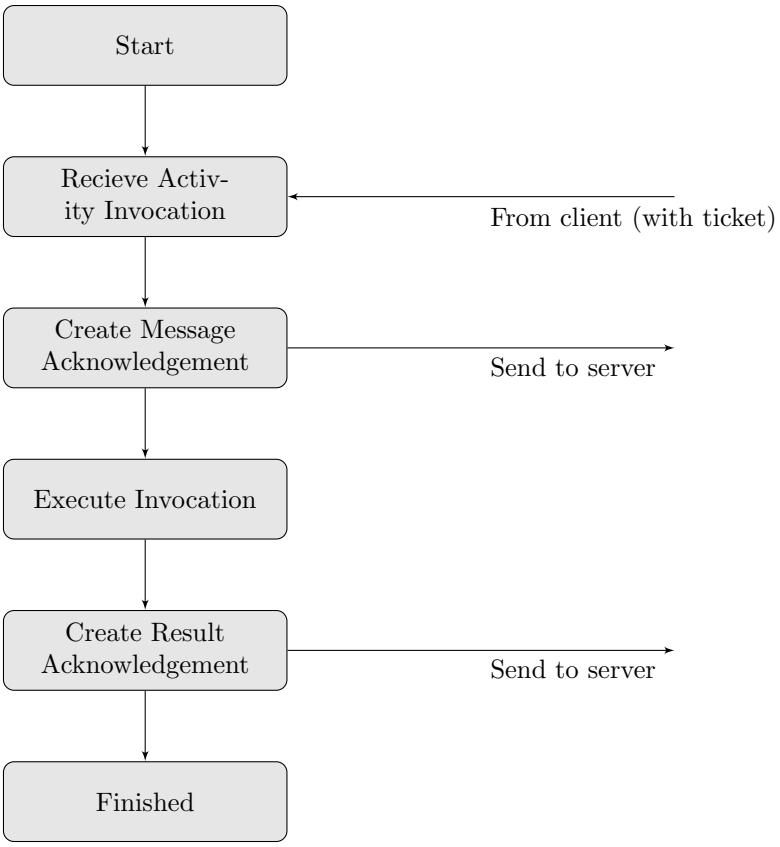
3.3 The messaging specification model in 0.0.2

These three flow charts shows the specification for the message routing between the client, the server and the zombie when the user invokes an activity on the zombie.
Note: Due to the implementation of SignalR (the transport) we will always get an acknowledgement whether what we send to the server is received or not.

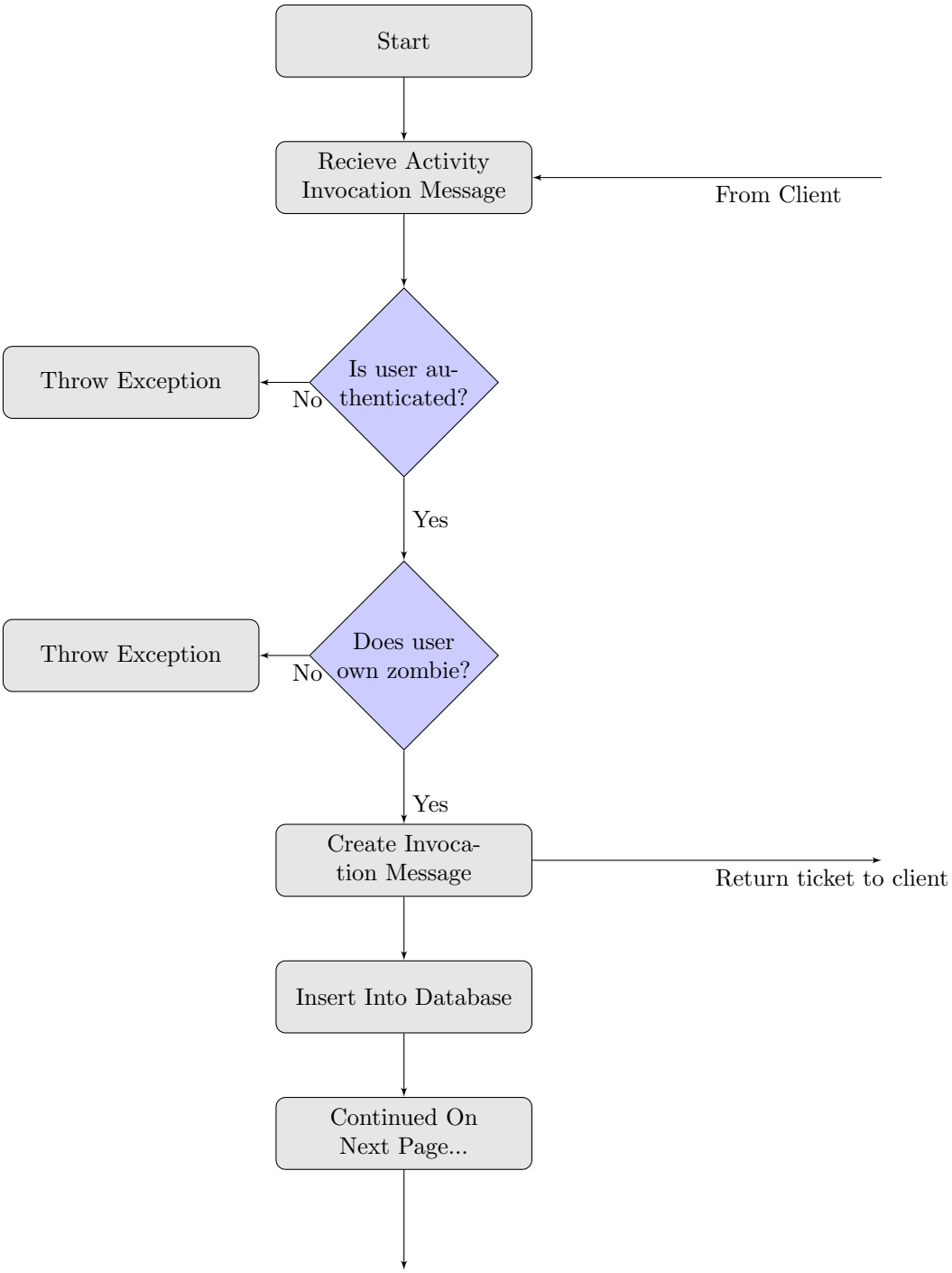
The Client

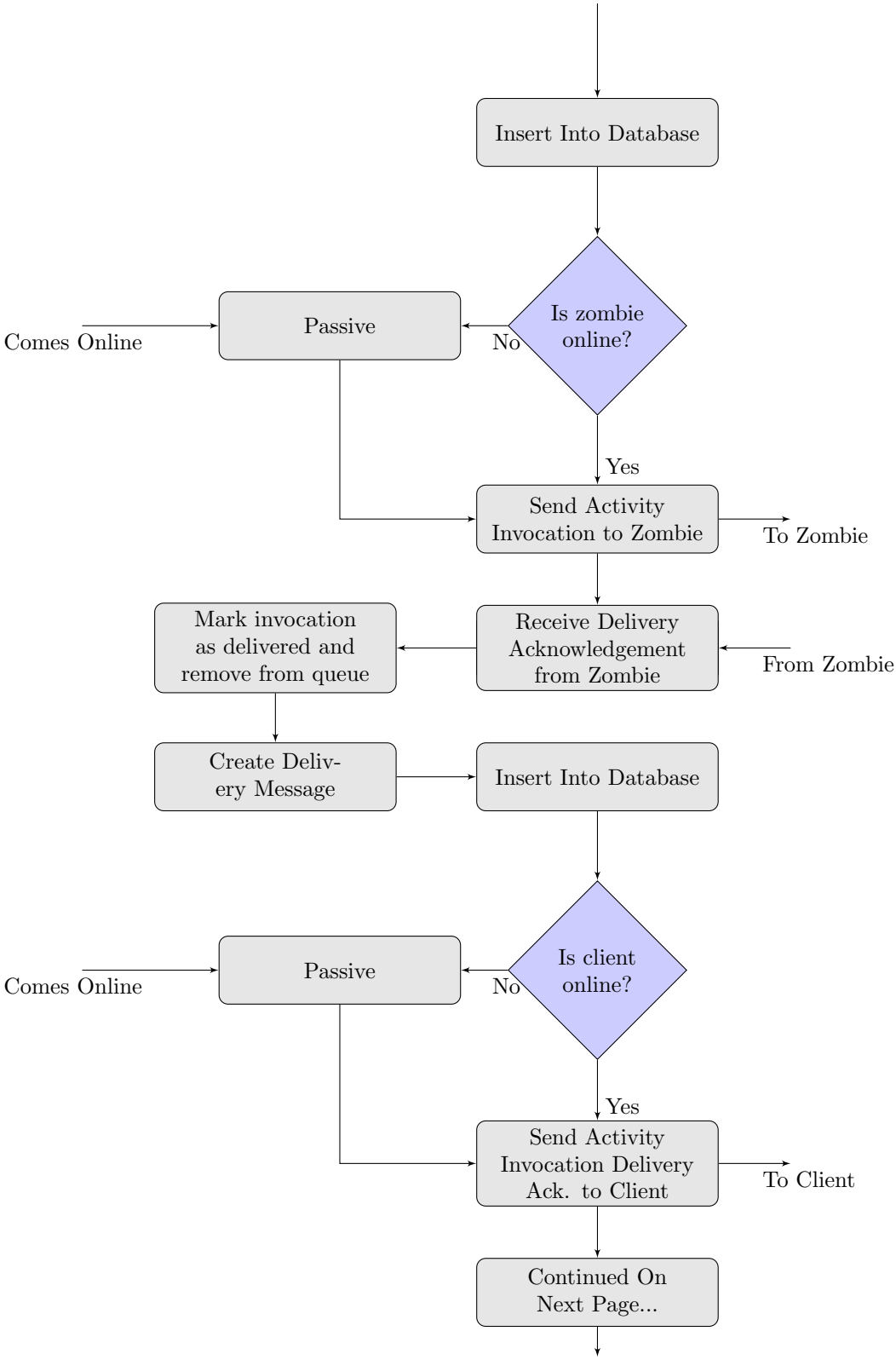

Start
Create Activity Invocation
Send to server (recieve ticket)
Passive
Invocation Delivered to Zombie
Create Message Acknowledgement
Send to server
Passive
Recieve Result
Result Response from Zombie (with ticket)
Create Message Acknowledgement
Send to server
Finished

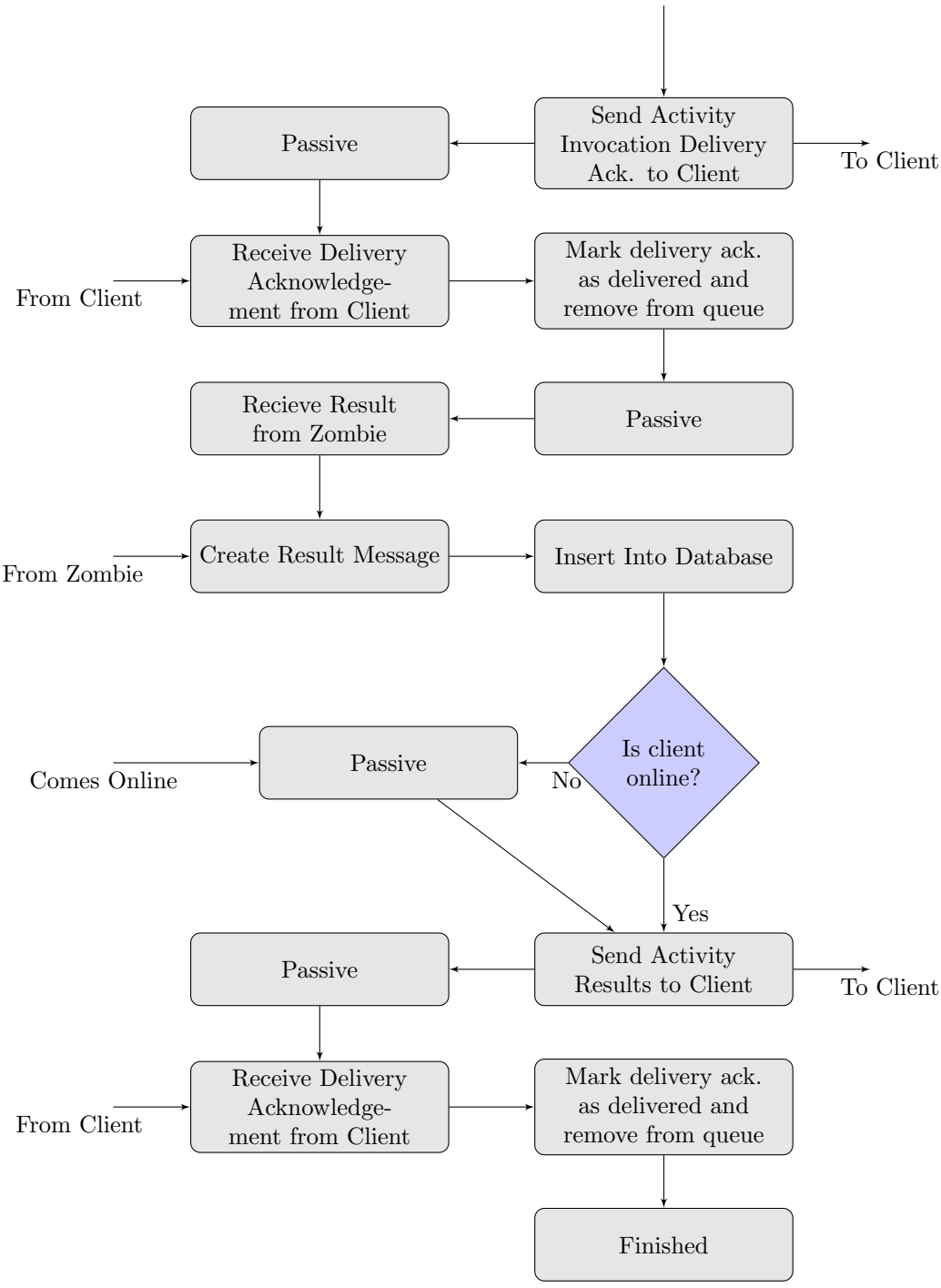
The Zombie


Start
Recieve Activity Invocation
From client (with ticket)
Create Message Acknowledgement
Send to server
Execute Invocation
Create Result Acknowledgement
Send to server
Finished

The Server


Start
Recieve Activity Invocation Message
From Client
Is user authenticated?
No
Throw Exception
Yes
Does user own zombie?
No
Throw Exception
Yes
Create Invocation Message
Return ticket to client
Insert Into Database
Continued On Next Page...


Insert Into Database
Is zombie online?
No
Passive
Comes Online
Yes
Send Activity Invocation to Zombie
To Zombie
Receive Delivery Acknowledgement from Zombie
From Zombie
Mark invocation as delivered and remove from queue
Create Deliv- ery Message
Insert Into Database
Is client online?
No
Passive
Comes Online
Yes
Send Activity Invocation Delivery Ack. to Client
To Client
Continued On Next Page...


Send Activity Invocation Delivery Ack. to Client
To Client
Passive
Receive Delivery Acknowledge- ment from Client
From Client
Mark delivery ack. as delivered and remove from queue
Passive
Recieve Result from Zombie
Create Result Message
From Zombie
Insert Into Database
Is client online?
No
Passive
Comes Online
Yes
Send Activity Results to Client
To Client
Passive
Receive Delivery Acknowledge- ment from Client
From Client
Mark delivery ack. as delivered and remove from queue
Finished