# 期中專題報告

## (一)專題內容製作動機與預期目標

在這個每個人手機上都有著社交軟體的時代，在使用中時常會看到社交機器人的蹤影，但是雖然人人都看過社交機器人但是卻不能保證每個人都可以開發出社交機器人。現今的機器人框架對開發者來說雖然有著各自的特色，但是因需求的多變也難免有所缺乏，這使的開發者在選擇框架上可能會有些困難。因此我們決定透過統整現在所知框架的特色藉此開發一個機器人框架，此外這個框架還可以彈性的擴充功能，讓開發者可以任意地自行增加功能、開發理想中的機器人。

在開發機器人初期選擇框架時總會遇到選擇上的問題，例如(1)程式碼複雜難懂使新手使用者難以理解、(2)需要自行建立或購買伺服器、(3)提供的的平台太單一又或者(4)該框架沒有支援或沒有提供想要的功能等，綜合上述我們期望開發的機器人框架可以解決以上在選擇框架時帶給使用者的問題。

本計畫將開發出一個框架用來解決上述問題，在本計畫完成後，該框架預期將擁有以下特點:(1)該框架架構重構性高:可以彈性的擴充使用者想要新增、使用的功能，不需大量修改程式碼(2)提供跨平台功能:提供一套可以統整各個平台的API，該API可以轉換各個平台的訊息格式再透過這套API轉發到其他指定的平台，讓使用者可以透過機器人串接各個平台，達成跨平台的功能。(3)使用者撰寫難易度低:本框架程式碼架構簡單明瞭，一個類別專注在單一功能上，使用者可以輕易地上手開發屬於自己的機器人。(4)運用NLP技術來分析語句，使機器人可以靈活應對訊息。

社交機器人的普及正加速來臨，尤其在內容重複性高、資訊容易被複製的領域，例如客服應答、抽獎文回覆等場景，導入聊天機器人(Chatbot)再適合不過了。現今有許多通訊社交軟體都提供了讓程式開發者自由建置機器人的服務；而對一般大眾而言，只要隨時拿出手機、電腦，進入平時聊天用的APP，就可以享受機器人的服務，不需要額外安裝特定APP，非常方便。

隨著社交機器人已經被廣泛地被使用在許多社交應用程式的開發中。許多社交平台都已經發表了自己通訊軟體的API，如Fb的Messenger Developer、Line Developer、Slack API等，許多機器人框架(Bot Framework)也已經被開發出來，但是隨著大眾的需求多樣化這些框架提供的功能也許會無法一一滿足，因此我們試著以此研究計畫來解決此問題，透過收集各個框架的特點打造出可以任意擴充功能的機器人框架(Bot Framework)。

### 1.建立一個可以彈性擴充任何功能的機器人框架

大部分的機器人框架雖然提供完善的功能，但客戶的需求可能會隨時間而有所變化，所以框架會需要改版，勢必會頻繁地更動程式碼，為了避免有牽一髮動全身的問題，因此確保軟體的重構性對於日後開發者能在短時間內新增功能是很重要的。如此一來可以降低長期頻繁修改造成的複雜性，並且增加程式可讀性以及軟體生命週期。

### 2.提供跨平台功能

現今市面上有發行許多不同通訊軟體，用戶可能會因情況的不同而同時使用多種社交軟體，但如果機器人只能與單一平台上的用戶做溝通，那整體來看是非常狹隘和不方便的。因此我們提供跨平台功能，不僅能使多平台間的資訊流通更為方便，也能建立一座橋梁成為平台間相互溝通的幹道。

### 3.情境式對談

我們使用NLP來分析接收到的語句，並提供擬人化的情境互動，使機器人更具有智慧，在回覆時能夠針對某個特定的情境有相對應的回覆，讓對談者感覺自己就像是跟真人講話一樣。

表1 各框架優缺點比較分析

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **chat bot** | **支援平台** | **優點** | **缺點** | **是否能跨平台** |
| Manychat | Messenger | 因為no code所以建立機器人簡單 | 單一平台。因為是no code所以缺乏多樣性 | No |
| Microsoft Bot Framework | Skype、kik,Messager、Slack、twilio、Line、email | 支援多語言的轉換，多平台。 | 原始碼複雜,難懂。因為是以多平台為1優先考量，所以並沒辦法將某些平台的特色發揮出來。 | Yes,透過bot service |
| IBM Watson Conversation | Slack、Kik、Messenger、Snapchat、Telegram、Twitter | 資料儲存在雲端上由IBM保護較為安全。 | 對於第三方平台的連結上有受限。 | No |
| Dialogflow | Line、Twitter、Slack、Messenger、Cortana、Alexa | 不須具備程式能力操作網路介面即可。 | 對答較為死板，可能無法得到預期的回覆。 | Yes |
| Amazon Lex | Messenger、Slack、  Twilio | 可與AWS平台上的服務做整合，提供強大的功能 | 訓練語言只支持美式英語，不支援他國語言。 | No |

## (二)整體時程與工作規劃

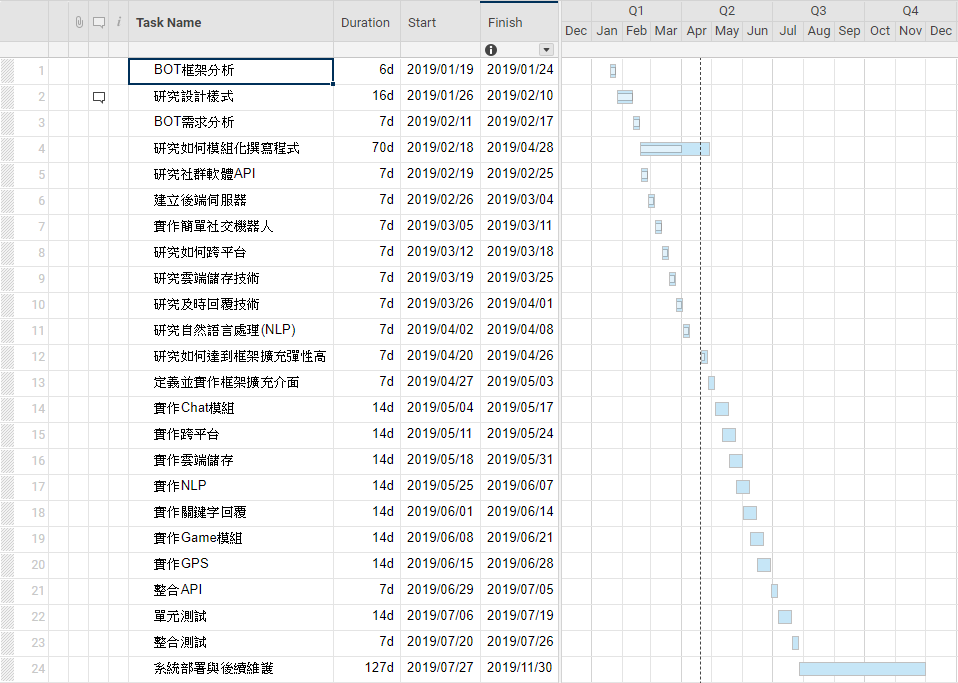


圖1 專題進度甘特圖

## (三)經費規劃及使用情形

基本上都是免費軟體開發，目前尚無經費需求

## (四)問題及如何達成預期目標

為了達成前述的研究動機並結合現有的機器人框架優點，我們預計於所提出的機器人框架上的功能包含(1)跨平台功能(2)Bot的語意分析與回應(3)小遊戲，其中(2)、(3)為包含在Plugin Storge中的plugin。

統整上述所提到之功能，我們將以圖2的架構來進行整合及詳細說明

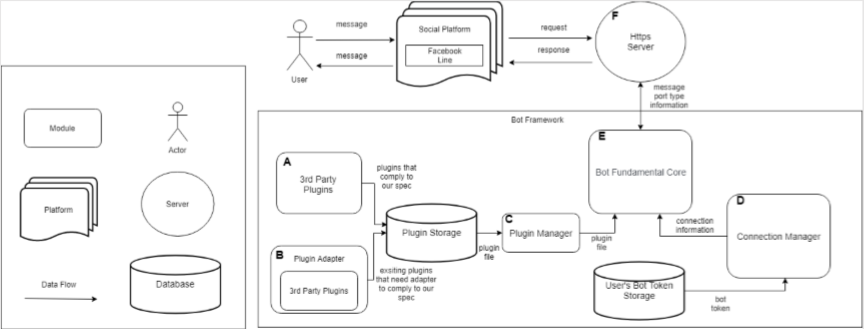


圖2 框架架構圖

A. 3rd Party Plugins:第三方插件，為機器人框架使用者要使用的插件的基本檔案，使用者需自行撰寫或者使用開源程式來編寫插件

B. Plugin Adapter:插件轉接器，可將第三方插件符合我們制定的插件規範

C. Plugin Manager:插件管理器，將所有插件統一管理，使用者可以以此來統一載入以及瀏覽所有插件

D. Connection Manager:Bot用來對社交平台(Facebook,Line)進行基本連線的管理，例如Port的監聽、平台的認證、Webhook連線以及傳送API

E. Bot Fundamental Core:此框架的核心模組，可以辨識來源平台，並透過插件管理器使用任何插件資料夾裡的插件來彈性的擴充想要增加的功能

F. Https Server:透過Https傳輸協定建立安全連線，並以此來傳輸、接收訊息

本研究的功能性需求將逐一講解

1. 跨平台功能(Cross-Platform)

本研究欲建立一套API，藉此可以對各個平台的API進行統整，圖3為跨平台架構圖，使用者對使用的社交平台發出訊息，經過社交平台上的API轉換格式，回傳給機器人，而機器人在回覆訊息前會先經過統整API(uniform API)，轉換成符合使用者欲傳送平台的格式，最後再將合法格式的訊息傳給使用者。

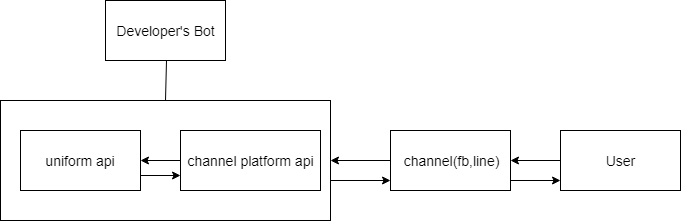


圖3 跨平台架構圖

在本研究中需要探討(1)平台api的訊息結構:圖4為fb的訊息結構，其中"recipient"代表接收者的編號"id"，"message"代表傳送的訊息格式"text"、內容。



圖4 Facebook Send message 格式

(2)將所有支援的社交平台API相似的部分整理並統一:根據步驟一的訊息整理，將API相似部分統整出來。例如各個平台訊息傳遞皆會包含送訊息者的訊息格式或內容等等，在統一訊息結構中便可以加入接收者編號、傳送者編號、訊息型態與訊息內容，(3)其中社交平台API不相似的部分例如接收、傳送者編號等資料在不同平台也會不同，這邊我們決定設立一個資料庫專門進行不同社交平台之間資料的蒐集與分析[13]用來統一資訊，例如:使用者id在Line與FB可能會不同，這時候在我們的資料庫內會給這個使用者一個專門在這個資料庫內進行身分認證的使用者編碼如圖5。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **platform** | **user\_id** | **nickname** | **group\_name** | **friend** |
| line | Ub3ba83cd4e5575b65494fdc28f521cae | John | 群組1 | Jake |

圖5 跨平台使用者資料庫範例

使用Node.js MySQL模組作為使用者資料庫管理、儲存，藉由欄位之間的關聯性來搜尋以及使用用戶相關資料

2. 插件式開發(Plugin)

由主系統+插件構成，主要定義為在不改變系統主體的情況下進行功能的擴充。程式撰寫者可以預先定義好系統擴充功能插件的接口，插件開發者可以根據此接口格式編寫插件來為系統增加功能，且獨立於主系統外的插件可以更易維護以及重複使用。圖6為插件系統架構圖。

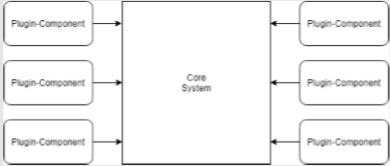


圖6插件系統架構圖

一個可以被重複使用的插件所需要以下的條件:

1. 插件本身的作用域與執行當下的作用域是互相獨立互不相干 的，也就是插件內部的變數不能影響當前使用者的執行環境的變 數作用範圍
2. 插件API的設置，使用者可以透過API來使用該插件功能
3. 插件需要一個可以統一管理、使用的接口(manager)
4. 依照各個插件定義每種不同插件的內容跟觸發條件來載入插件

3. Bot的語意分析和回應

為了讓Bot能達到更為貼近我們真人的回應，我們能使用NLP來處理並分析使用者的語句。NLP系統流程可分為四個主要流程步驟:

**分/切詞(Tokenization)->詞性標註(POS Tagging)->語意組塊(Chunking)**

**->命名實體標註(Named Entity Tagging)**

分/切詞:每種語言的詞彙和語句結構都不同，以中文和英文來說中文不像英文那樣有空格來分詞，所以在分析文字前就必須把一個個漢字分解成合適的詞語。

詞性標註:詞性是詞彙的語法屬性是連接詞彙到語句的橋樑，而詞性標註就是來分辨出一個句子中每個詞所扮演的語法角色。

語意組塊:將標註好詞性的句子按句法結構把某些詞聚合在一起形成比如主語、謂語、賓語。

命名實體標註:命名實體識別用於識別文字中具有特定意義的實體，常見的實體主要包括人名、地名、機構名及其他專有名詞等。

以下我們舉個簡單的例子:

我看見小孩在草地上玩球

經過**分/切詞**之後變成:

我 看見 小孩 在 草地 上 玩 球

再來做**詞性的標註**:

我 看見 小孩 在 草地 上 玩 球

名 動 名 名 動 名

接著，按照主語、賓語來組合最後便能得出句子大概的意思:

我 看見 小孩 在 玩 球

主 動 賓 動 賓

本研究將建立一個語意彙集庫能供開發者新增自己所需要的對答句，並且我們將所建立的對答句分類(制定一種規則)，當使用者傳送一句語句時經NLP分析完語意後，並根據分類來決定回應句，例如:當句子中出現"預約、門診、看診"時，機器人就能根據此來提供相關的回應(門診表)，而機器人語意分析的過程，如圖7所示

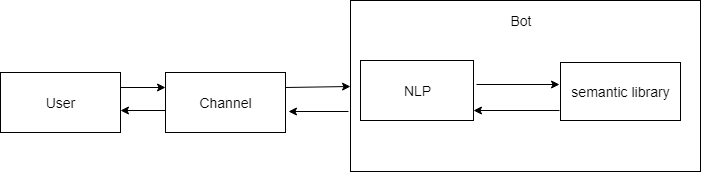


圖7 Bot語意分析架構圖

4. 框架擴充彈性高

擴充彈性屬於非功能性需求，為使功能可以快速且方便的維護與擴充，除了規定統一的Coding Style之外，還可以從前面所提及的程式碼層次來下手，也就是程式碼重構，例如：

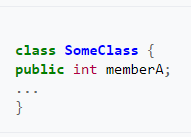
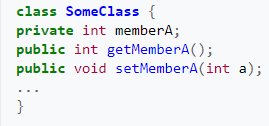
 

圖8 封裝成員變數(重構前) 圖9 封裝成員變數(重構後)

適當的封裝也是重構的一種，可以將物件使用介面的程式實作部份隱藏起來，不讓使用者看到，同時確保使用者無法任意更改物件內部的重要資料。它可以讓程式碼更容易理解與維護，也加強了程式碼的安全性。重構最重要的一部分即是修改後的驗證，為了確保原來的功能沒有被改壞，必須透過不斷的測試來確認。

另外，增加程式的可讀性也能夠使軟體易於擴充，利用模組化的概念降低耦合度，例如：

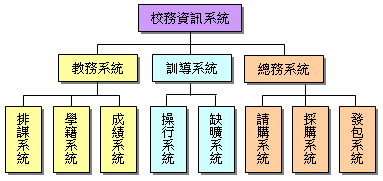


圖10 校務資訊系統

圖10將各個系統分門別類，以達到模組化。然而除了模組化之外，還要限制模組之間的關聯性，減少它們的複雜度。如圖10中的成績系統的計算方式可能會需要缺曠系統以及操行系統的參與，如果讓它們可以直接互相呼叫(成績系統直接呼叫操行系統和缺曠系統)，會使複雜度提升，但若改成統一透過教務系統與訓導系統(成績系統透過教務系統呼叫訓導系統，訓導系統再個別呼叫)來建立溝通的管道，如此一來模組之間的複雜度就會降低。

5. 轉接器模式(Adapter Pattern)

我們使用轉接器模式將插件(plugin)的接口轉換成符合目標系統的介面，使兩個不同的系統、程式介面由原本無法配合一起工作，透過此模式可以將想要使用的介面轉接成想要的系統介面，藉此來符合目標系統的執行介面，以下是圖11轉接器模式的示意圖。

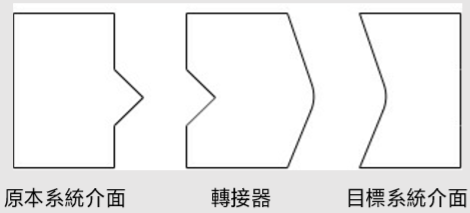


圖11 轉接器模式示意圖

## (五)成果

下圖12為簡單的跨平台接收訊息範例，詳情請見

<https://github.com/Mist-Rain/Bot-Framework/blob/master/README.md>

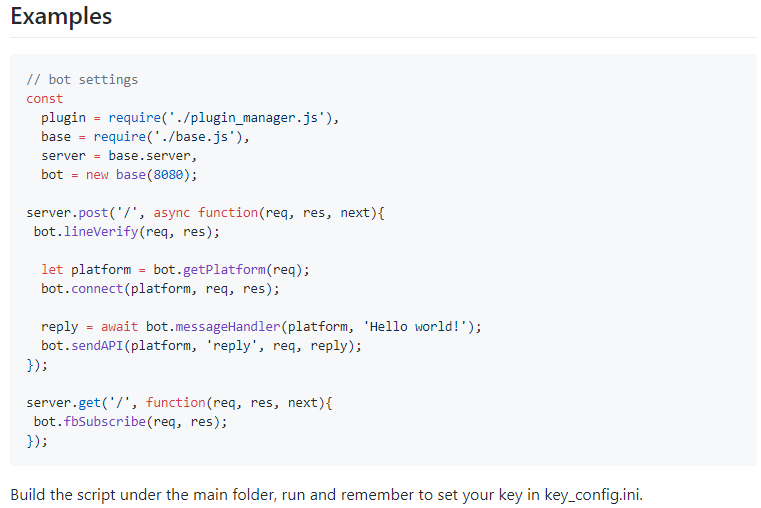


圖12