

**Chat-Bot with Machine Learning**

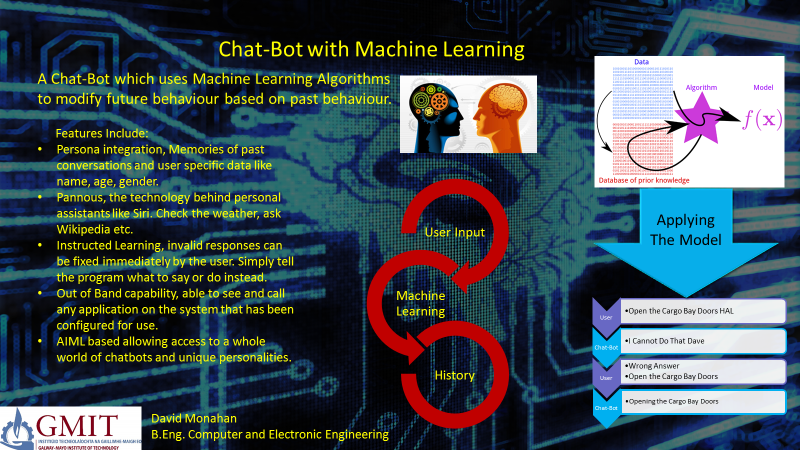
**David Monahan**

Bachelor of Electronic & Computer Engineering Galway-Mayo Institute of Technology

2016/2017

Supervisor: Paul Dunne, Galway-Mayo Institute of Technology







This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering in Computer & Electronic Engineering at GalwayMayo Institute of Technology. This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.



I would like to acknowledge the good people at Stack Overflow (<https://stackoverflow.com/>) without whom I would still be trying to make this thing work.





A Chat-Bot is a program which attempts to emulate conversations with a person. The aim of this project was to create a Chat-Bot which could learn new responses and modify its existing responses based on user input. To that end I decided to create a virtual assistant style of Chat-Bot in line with commercial offerings like Siri.

This project uses a heavily modified form of Program AB (AIML 2.0 reference implementation), and the AIML(Artificial Intelligence Markup Language) 2.0 compliant S.U.P.E.R (Substantial Upgrade over Previous Example Robot) bot. The completed Chat-Bot does not exhibit all of the behaviours and features that I had intended to add but does include learning based on Predicates and direct user Instruction. A selection of external API's allow the Chat-Bot to fulfil its role of personal assistant and go beyond merely simulating conversation.





* Title
* Poster
* Declaration
* Acknowledgments
* Summary
* Introduction
* Architectural Block Diagram
* Body of the Project
* System Integration
* Conclusion
* References





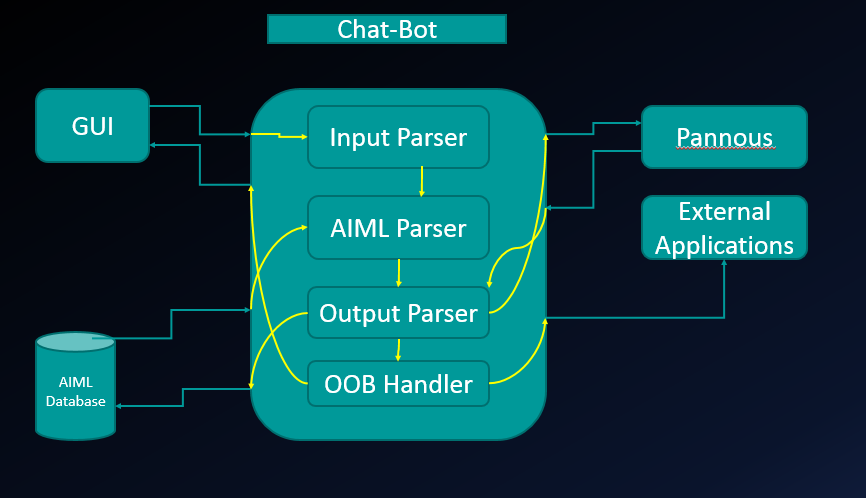
A Chat-Bot is a program designed to emulate person to person conversation. The applications of such a program are wide ranging, from simple toys intended to amuse or entertain or as part of a larger user interface designed to simplify interaction with users who may not be very tech-savvy. Chat-Bots are becoming more popular in recent years with the rise of applications like Siri and Cortana becoming a novel way for users to interact with their mobile devices or PCs.

Chat-Bots have a long history going back to ELIZA in 1966 created by Joseph Weizenbaum. ELIZA is considered the first Chat-Bot and follows a rules-based model. Rules-based Chat-Bots generated their responses according to strict rules, and are the easiest to make but have the narrowest range of responses as a result. ELIZA was designed to emulate a psychiatrist and would generate responses by repeating back to the user what they said under the pretext of wanting to know more about the subject or enquiring about how they felt about it. While very simplistic in nature ELIZA was notable due to its ability to fool several users into thinking they were talking to an actual person.

Since then Chat-Bots have expanded in both complexity and capability and while the ones available today are very advanced they still struggle when confronted with an open-scope conversation. Many Chat-Bots are designed with a specific scope in mind, i.e. the conversation can only be about certain things or contain certain questions. This means the rules for every possible response are easier to create but also limits the usefulness of the Chat-Bot.

Machine Learning then is an attempt to solve this problem. Pure Machine Learning involves the use of statistical analysis and Neural Net Processing(NNP) to create responses to a given input from a set of data. While such Chat-Bots do exist, Microsoft and Google have both produced quite advanced versions in recent years, their output is often an incomprehensible mess. This is due to the Bots brain only recognising patterns in words and sentences but not being able to comprehend the meaning behind those words. The solution then is to add rules to control how it learns.





The crux of this project consists of a GUI to allow the user to interface with the application, an AIML database to store all the patterns used to define and create responses and a set of APIs to allow the application to access external utilities like Pannous, Firefox etc.

Internally the program consists of an Input Parser which cleans up the users input into something that can be matched against the AIML Parsers selection of AIML patterns (over 50,000 included in this project). The output of this is handed off to the output Parser and OOB Handler which determines the correct response for the Bot to take.



This project consists of three major components. The S.U.P.E.R AIML 2.0 Bot, Program AB AIML 2.0 reference implementation, and the GUI.

**S.U.P.E.R. AIML 2.0 Bot** [1]**:**

The S.U.P.E.R. AIML 2.0 Bot or Substantial Upgrade over Previous Example Robot Artificial Intelligence Markup Language Bot is the knowledge base for the Callmom virtual assistant[5]. This is a set of AIML 2.0 compliant files which contain the Bots “brain”. This is essentially a large database of patterns and responses to be used by the Chat-Bot during normal operation.

AIML or Artificial Intelligence Markup Language is an XML (eXtensible Markup Language) compliant standard for creating and storing Chat-Bot patterns and responses. AIML was created by and is actively supported by the ALICE AI Foundation. The current version of AIML (2.0) was launched in 2013 and contains many new tags specifically for virtual assistant style Chat-Bots.#

AIML tags and layout:

<aiml>: begins and ends each AIML document.

<category>: The base “unit of knowledge” for an AIML bot. Each category must contain exactly one pattern but can have any number of templates.

<pattern>: The pattern to match to the users input. This pattern can be simple plaintext, contain wildcards or even contain substitutions based on predicated values such as the users name.

<template>: This is the template used to create a response to the matched pattern. This can also contain wildcards, or substitutions. Multiple templates can be included in a category using sub tags to indicate how to select the correct template for the situation. For example <random></random> tags indicate to pick a template at random.

<srai>: symbolic reduction or symbolic recursion artificial intelligence. This is used to redirect to a different template within the same Bot. Usually to reduce inputs with common responses down to the same response. For example Hello, Hey, Hi, etc. can all be redirected using <srai> to use the same template.

The S.U.P.E.R. Bot is designed specifically for mobile virtual assistant applications and contains patterns to match standard virtual assistant requests such as Google search and calls to launch external applications. This is handled through the use of <sraix> and <oob> tags.

<sraix>: This is an extended version of <srai> used to query other bots for a pattern and use their template to generate a response.

<oob> Used to indicate that the pattern should launch and Out Of Band request. The response will contain an xml formatted pattern wrapped on <oob> tags and will have to be processed by the output parser in the application. [4]

The S.U.P.E.R. is fully compatible with Program AB and can be extended using the superbot SDK available from the ALICE Foundation and used for creating commercial grade Chat-Bots and Personal Assistants.[6]

**Program AB AIML 2.0 Reference Implementation**[2]**:**

Program AB is a Java application intended as a reference for implementing the new features available in the AIML 2.0 draft specification. It includes support for the new <sraix> tag allowing the bot the connect to the internet and query other bots for responses to a given input and also allows for the addition of new tags. It also allows for the use of sets and maps and wildcards. A notable improvement over AIML 1.0.

The version of Program AB being used in this project is a fork of the original beta implementation launched by the Alice Foundation in 2013. This version keeps the original AB.Jar as a library and uses a maven build file (pom.xml) from Hendy’s lumenrobot, itself a fork of Program AB, to rebuild the AB.jar with new source code to allow the modifications required for this project to be made.[7]

Program AB contains the brains of the Chat-Bot. This is all of the parsers for loading the AIML files, parsing the categories and then matching the user inputs to get the best response. Program AB can also connect to the internet to check for other bots having a pattern that matches using <sraix> or to connect to various internet APIs like Pannous. However these do not work out of the box and hence the maven build file required to modify and extend Program AB to allow extra functionality.

Program AB also includes an API handler for matching Pannous [8] patterns in the AIML but the API implementation that comes pre-build with Pannous is out of date with the current Pannous API. This had to be updated and have its API key and login modified. This code had to be recompiled into the AB.jar using the maven build file pom.xml.

**The Graphical User Interface:**

This contains the brunt of the work done on this project. Program AB is a command line only application and even then not a very friendly one. To alleviate this problem and to allow the application to give a much more friendly user experience a GUI was created as a wrapper around the programs CLI interface. This allows the user to chat away without having to worry about control characters on the input and allows the outputs to be much more interactive. For example image links are now automatically resolved and will display an image instead of just a plaintext html link.

The inputs sent to Program AB have to be in a strict format i.e. no capital letters, no xml control characters (<,>, etc.) no websites or e-mail addresses. This means all inputs have to be carefully worded. The GUI gets around this problem by using a large list of substitutions to clean up the inputs and replace any problematic characters with html style substitutions. E.g. < goes to &lt; These substitutions are reversed on the output to hide this functionality from the end user and improve the user experience.

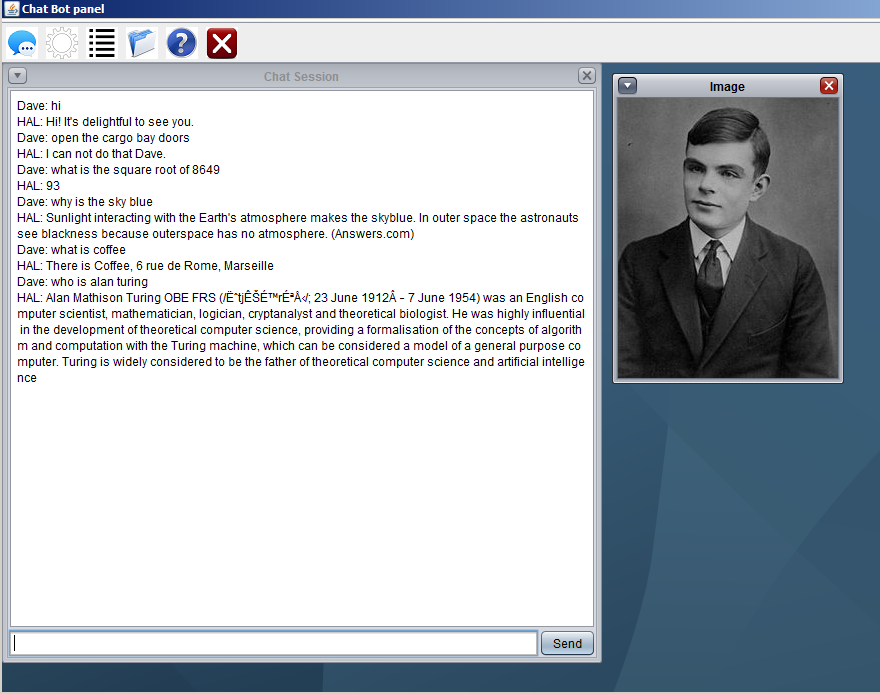
To implement the GUI required the Main class for Program AB to be stripped and rewritten. The new Main will setup Program AB to use the AIML located in a relative folder to the application and then configure all the necessary paths and properties of the Bot relative to this location. This makes the Bot a bit more stable and prevents it from crashing due to a misplaced file in the middle of a chat-session.

The GUI allows for the user to modify the bots properties and predicate settings. This allows the user to set custom knowledge for the Bot on start-up. This can be useful for creating multiple custom Bots for different users or for testing how the Bot learns if given false initial information.

The GUI includes an Out Of Band handler for parsing outputs containing <oob> tags and launching the appropriate local application for the user. The GUI includes options allowing the user to set the path for each supported <oob> pattern to a preferred local program. For example not everyone uses Firefox as their browser of choice.

The main Chat-Bot window is loaded into a Desktop as part of the GUI. Program AB loads all AIML currently available into memory on start-up. By having the main Chat-Bot display on an internal window this allows the user restart the Chat-Bot and reload AIML from a different Bot without having to restart the entire application.

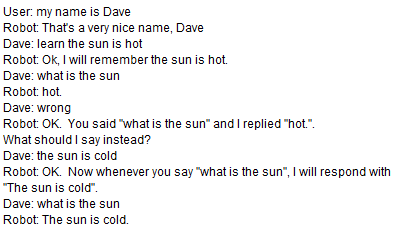
The GUI facilitates conversation with the Chat-Bot in a much smoother and more intuitive environment than the base CLI by streamlining the input and output parsing process and hiding this from the user.



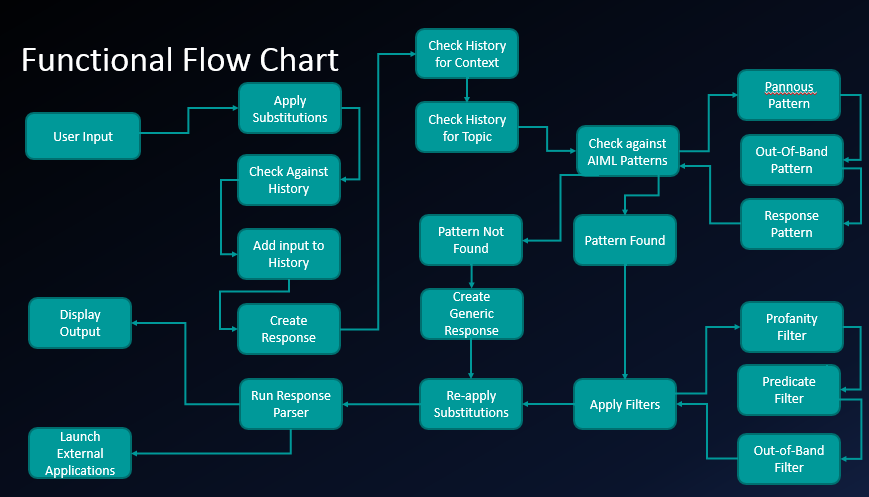


The entire project then is an attempt to take an existing set of patterns and templates (S.U.P.E.R) a program for parsing those patterns (Program AB) and try to make a usable product out of them with some learning abilities added in. In this I feel I have succeeded.

The application contains the following features:

1. Predicated and Instructed Learning. The Chat-Bot can learn information about the user through conversation by matching predicated patterns like the users name, address, age, and location. The Chat-Bot can also be instructed to learn something by the user explicitly telling the Bot or by the user correcting the Bot on how it responds. E.g  
   
2. Full Pannous API Implementation. Pannous is the technology behind Chat-Bots like Siri and is used for expanding the capabilities of the Chat-Bot into the Personal Assistant Realm. Allows the Chat-Bot to respond to questions outside the scope of its knowledge or to get up to date information on a topic from the internet. E.g. Whats the weather like in Galway? Who is the current Irish President?
3. Out of Band Handler to handle launching of external applications based on a user input. Some applications like the Web Broswer can also be passed additional parameters based on the user input to expand this functionality. E.g. can you google gmit?
4. Some smart patterns allow for the appearance of Intelligence. For example asking the Bot for the current Data will use the Java Data class to return an accurate response.

A breakdown of how the application works in Chat-Bot mode is shown here:





Overall I am quite happy with how the project turned out. Some learning functionality that I wanted to implement has been left unfinished as I was not able to get it working on time. The AB.java class contained in the AB.jar file within the project contains experimental methods for analysing chat logs and generating AIML files from the results. Unfortunately while this would be very impressive if it worked I was unable to get this class to produce anything other than gibberish.

However the main goal of the project was to create a Chat-Bot with the ability to learn from the user and in that I feel I have succeeded. Chat-Bots with learning abilities tend to suffer from the following problems:

* Lack of persona, gives conflicting answers to questions like where do you live? What country do you live in? What city do you live in? This is due to how the program learns this information from analysing conversations. For each question only the most common answer is returned. My program avoids this by being able to recognise common information and adjust the predicated values to match.
* Grammatically flawed responses. Neural Net Chat-Bots use statistical analysis of conversations to try and emulate a conversation through patterns alone. Because the computer does not understand the meaning behind words only the patterns in which they are used the resulting output is something that almost correct but not quite. By having only grammatically correct patterns available to choose from this program will always at least give a intelligible answer even if the answer is not what the user wanted.
* Limited scope of conversation. Chat-bots, especially rules-based models like this one tend to be very limited in the range of conversation that can be held. Since only patterns that the Chat-Bot has been programmed with can be used the user is expected to stay within that scope. My project is not perfect in this regard but with the ability to extend the Bot with new AIML as it is created and with the user’s ability to correct the Chat-Bots responses this problem is alleviated somewhat.
* Unknowable information. Some information that might be considered common knowledge to people such as the current president or what the weather is like outside are completely alien to a computer. By giving the Chat-Bot the ability to search out answers to questions via Google and Pannous the Chat-Bot can give the illusion of intelligence. After all a Chat-Bot that just responds I don’t know to every question doesn’t appear very intelligent.

There is of course always room for improvement, in the future voice recognition, text-to-speech and statistical analysis based learning could be added to the project without changing any of the core functionality.

The project as it stands is something I feel proud to have built. It looks good, works well, is very very stable (over 6 weeks since the last crash) and is something I will likely continue to improve in the future.

**References:**

[1] <https://code.google.com/archive/p/aiml-en-us-foundation-super/>

[2] <http://alicebot.blogspot.ie/2013/01/program-ab-aiml-20-reference.html>

[3] <https://github.com/lumenrobot/program-ab>

[4] <http://callmom.pandorabots.com/static/reference/#elements/-lt-oob-gt->

[5] <http://callmom.pandorabots.com/static/callmombasic/>

[6] <http://www.alicebot.org/superbot.html>

[7] <https://github.com/lumenrobot/program-ab>

[8] <http://pannous.com/>