Suncoast Computer Science

Senior Project

SIA:

Socio-Cognitive Intelligence Agent

Author:
Terrell Ibanez

Supervisor:
James Krumenacker

2012 - 2013

Creative Commons License

With the understanding that:

- *Waiver:* Any of the above conditions can be waived if you get permission from the copyright holder.
- *Pulic Domain:* Where the work or any of its elements is in the public domain under applicable law, that status is in no way affected by the license.
- *Other Rights:* In no way are any of the following rights affected by the license:
  - Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations;
  - The author's moral rights;
  - Rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.

Contents

Section A: Analysis

A.1: Problem Analysis

With the continuing increase in the capabilities of mobile devices, comes the problem of keeping the user interface simple, without sacrificing functionality. Adding more buttons and menus only works up to a certain point, and this has resulted in the development of new ways to interact with a system, such as predictive text, word paths, and voice recognition. Siri, one of the most recent innovations in human-computer interaction, is arguably the first voice-recognition-based input designed well enough for practical use. However, it is not a perfect implementation, the major drawbacks being both its inability to refer to former subjects on context, and its tendency to suggest a web search as opposed to simply pulling from a data source, especially when a desire to perform a web search was not implied.

For example, given the query "What is ..." , Siri will respond with "Would you like me to search for..." , as opposed to the requested information. Every single time this occurs, it loses the human-like impression it has to the user, reminding them of the fact that it is a machine, bound by the limitations of what it can understand. While a competent AI is the ideal solution to the human-computer interaction problem, there is another way to approach it.

Contrary to a full-fledged AI like Siri, one of the earliest examples of natural language processing, the ELIZA (DOCTOR script) program, was

much more successful at convincing users that it was human, due to its use of pattern recognition and social engineering. The problem of today's AI is the difficulty of covering all of the varied use cases, as machine learning typically does not easily handle the exponential difficulty of multiple factors. A good example of this is autmated driving, the numerous situations derived from varying conditions. Weather, other drivers, street signs, stop lights, and pedestrians are just a few of the many possible factors affecting driving behavior.

When considered, real AI is more of an academic objective, as in the buisness world, the backend is unimportant to the end user, because the interface is the only thing they interact with. That said, the only thing the user is actually looking for, in the case of Siri, is a human-like intelligence agent. The typical end user does not care whether or not the backend is formed with a real AI that learns, as long as it is human-like, and gives an intelligent impression. Because the end goal is not actually AI, the backend can be replaced with a pattern recognition algorithm combined with a grammar model of english to achieve similar/better results than a real AI. This pattern recognition, with a proper model, can appear just as human, while being much less complex. ELIZA's implementation in BASIC is only 256 lines long.

A.2: Criteria for Success

For the program to be considered a successful implementation of the proposed solution to the aforementioned problem, there are two major goals to look at: Realism, and Usability.

A.2.i: Realism

There are multiple factors that affect a user's impression of a chatterbot, namely Comprehension, Variation, and Response Time. This has to do with cognitive dissonance, wherein the illusion of a computer being human is created through the two conflicting ideas. The human behavior conflicts with the knowledge that it actually is a computer, and forces the mind to resolve the conflict by changing the impression so that it is perceived as human. It is important to note that this only works as long as the human-like behavior exists, as cognitive dissonance works both ways. Once the behavior defects in any way, the mind abruptly shifts its impression back to that of a computer, which is difficult to come back from. It is much easier to convince someone that something is a computer, rather than a human.

A.2.i.a: Cognitive Process

The first major part of maintaining the illusion is by appearing to follow the human cognitive process with regards to information. Easier said than done, as not only does this require the agent to "understand" a question or command asked of it, but it should also be able to comprehend said question or command while including prior context. Humans may not have the best memory, but if they ask "When was he born?" right after asking "Who is Thomas Edison?", it is understood that "he" refers to Thomas Edison.

A.2.i.b: Variation

The second major aspect of human behavior is the variation of speech. There are multiple ways to express the same idea, and it is rarely expressed the same way each time. "Sounds good", "Okay", "Sure thing", are just a few of the many ways that one can acknowledge that information has been received and understood. One of the easiest markers of a computer is its tendency to

repeat the same response, when given the same input. This rarely happens in real conversations.

A.2.i.c: Response Time

A more subtle, but still important factor is Response Time. While a slight pause is acceptable (and possibly even better), a pause that takes too long will strike the user as un-human, and shatter the belief.

A.2.ii: Usability

An intelligence agent that replicates human behavior well, but does not provide assistance, lacks purpose. While ELIZA does much better than Siri at convincing the user, it doesn't provide particularly useful information.

A.3: Prototype Solution

The solution's UI layout as well as manner of operation will follow the current trend in intelligent agents, to place an activation button centered at the bottom of the screen underneath a transcript of the past queries and responses. The transcript is necessary to make the interaction between the user and the agent appear to be a conversation, while the activation button is placed specifically for ease of use. Placement at the bottom is to prevent the user from needing to shift his/her hand up the screen to activate, allowing for one-handed use. It is centrally placed for the same reason, only horizontally. While it is less likely that the user would have to shift his/her hand to the side, if placed differently, the central placement also removes the bias between left and right hands. The best examples of intelligent agents following this layout are Apple's Siri, and Samsung's S-Voice, as shown in Figures 1 and 2.

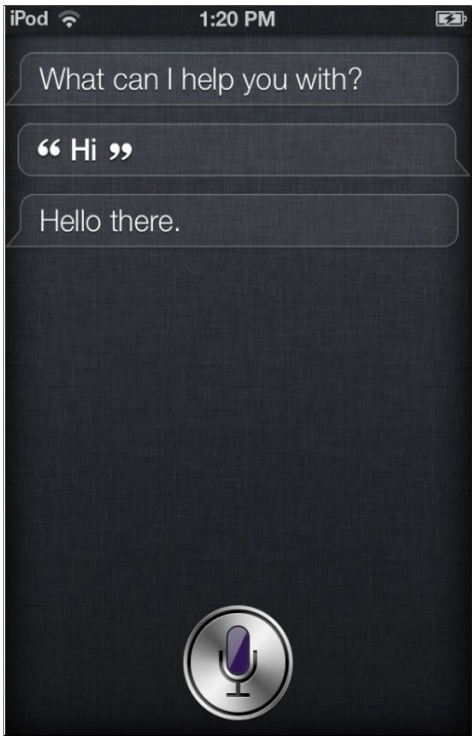

Figure 1: Apple's Siri

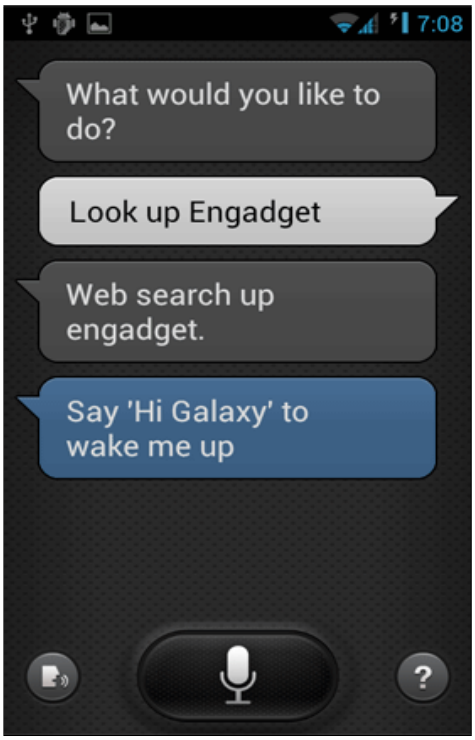

Figure 2: Samsung's S-Voice

Section B: Detailed Design

The design for the solution will utilize pattern recognition to classify words, clauses, and intended request, based upon an english grammar model specificially for intelligence agents.

B.1: English Grammar Model

Figure 3 shows the designed Grammar Model, expressed in EBNF (Extended Backus-Nour Form), where the " , " is the concatenation operator, " | " is the definition-seperator, " [" and "] " are the optional-symbols, and the " ! " is the exception operator. The grammar assumes that spoken sentences can be classified into one of three types: Request for Information(RFI), Request for Action (RFA), or Statement for Information (SFA). For the most part, only the first two are very relevant, so the designed model ignores SFAs. While there are relevant cases for SFAs this is only a simplified grammar model, which only aims to be mostly comprehensive, they can be added at another time.

B.2: Linguistic Decomposition

Linguistic Decomposition first starts by classifiying individual words into their types, extracting clauses from those words, and then determining the

```
<Interro>         = "who" | "what" | "when" | "where" | "why"
                  | "where" | "why" | "how" | "which";
<Action>          = "tell" | "show" | "describe";
<Copula>          = "is" | "was" | "are";
<Article>         = "the" | "a" | "an";
<Preposition>     = "of" | "for";
<Subject>         = ? !<Interro> | !<Action> | !<Copula> | !<Article>
                  | !<Preposition> ?;

<Pronoun-Clause> = <Article>, <Subject>;
<Action-Clause>  = <Action>, ["to"], "me", ["about"];
<Interro-Clause> = <Interro>, <Copula>;
<Subject-Clause> = <Artle>, <Subject>;

<Info-Request>   = <Interro-Clause>, <Subject-Clause>;
<Action-Request> = <Action-Clause>, <Subject-Clause>;
```

Figure 3: Simplistic English Grammar Model expressed in EBNF

intended information based on the clauses' relations to each other. The subject only affects the way the structure is analyzed if it's a special topic used idiomatically, the most common example of which is the weather. "What is the weather like today?" does not make sense (for this simpler model) because instead of referring to weather as a concept, the user is referring to the current weather status. Asking about weather as a concept can only be done by asking "What is [the] weather?"

Ideally, the linguistic model should be loose, but fill all known use cases. It is better for consistency to have false positives over false negatives, as a false positive does not hurt the human impression, and may actually improve it.

Algorithm 1 Word Classifciation

```
1: procedure ClassifyWord(Word)
2:     for all Types do
3:         if Word ∈ Type then
4:             return Type
5:         end if
6:     end for
7:     return "Subject"
8: end procedure
```

B.2.i: Pattern Matching

Algorithm 1 shows the classification of words according to their word type. If the word in question does not match one of the ones in the wordtypes we have predefined, it is usually safe to assume that said word is a subject. Once the words are classified, individual clauses are formed by first identifying the subject clause(s), and then identfying the remaining types based on the

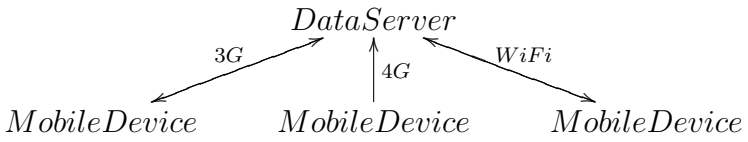

Figure 4: Server-Client Model for Mobile Devices

english model provided in Figure 3. The longest possible are checked first, to make sure a clause with optional parts includes those properly. This is demonstrated in Algorithm 2.

B.3: Modular Organization

One of the major drawbacks to mobile devices is their weaker hardware specs, as compared to a traditional laptop or desktop. To mitigate the effect of hardware differences, a server-client model was chosen, as shown in Figure 4.

The Mobile Device is not responsible for actually understanding and retrieving the information, only the speech recognition and Text-to-Speech (TTS). The server-client model is important as any updates made to the server affect it in real time, without the user having to update their end. This prevents fragmentation, and is much more convenient. The model can be improved by adding a cache, to relieve load off the data server, as well as any source servers. A cache, as shown in Figure 5, the cahce would also improve response times.

Of course, there are some major disadvantages to this approach. The largest of which being the inability for the client to comprehend any requests that only need to be handled locally, particularly commands.

Algorithm 2 Clause Classifciation

```
1:  procedure BuildClauses
2:      i ← 0
3:      mode ← "None"
4:      for all Words do
5:          if Word.Type = "Article" then
6:              Clauses(i) ← new Clause
7:              Add Word to Clauses(i)
8:              Clauses(i).Type ← "Subject"
9:              i ← i + 1
10:         else if Word.Type ="Subject" then
11:             if (mode ="Subject") and
12:                 (Clauses(i).Type ="Subject") then
13:                 Add Word to Clauses(i)
14:             else
15:                 Clauses(i) ← new Clause
16:                 Add Word to Clauses(i)
17:                 Clauses(i).Type ← "Subject"
18:                 i ← i + 1
19:             end if
20:         else
21:             if mode ="Subject" then
22:                 Clauses(i) ← new Clause
23:                 Add Word to Clauses(i)
24:                 Clauses(i).Type ← "Unknown"
25:                 i ← i + 1
26:             else if mode ="Unknown" then
27:                 Add Word to Clauses(i)
28:             else
29:                 Clauses(i) ← new Clause
```

```
30:                 Add Word to Clauses(i)
31:                 Clauses(i).Type ← "Unknown"
32:                 i ← i + 1
33:             end if
34:         end if
35:     end for
36: end procedure
```

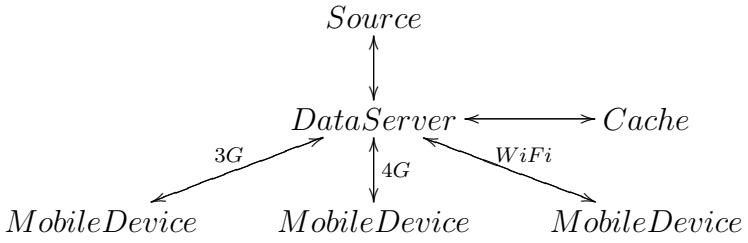

Figure 5: Improved Server-Client Model

Section C: Development

C.1: Implementation Specifics

PHP was chosen for the server side because of its place as the standard in web development. Android was chosen primarily because of its APIs, but also has the advantage of posessing the largest share in the mobile market, making it applicable to as many people as possible.

C.2: Testing Strategy

Considering the vast information the server is capable of retrieving, and the multiple ways a request can be formed, the ideal way to test is to check every possible sentence in the english language. However, this is far from practical. The easiest way to go about practical testing is to simply throw every type of request at it. Beyond the developer's testing, this is best achieved through a public beta, with users reporting bugs. Structured test cases can be built from these reports, and given enough time, will soon come to represent the most likely requests to be made.

C.3: Source Code

C.3.i: Server-Side (PHP)

The following shows the server-side code, written in PHP, with a representational state tranfer architecture. The server takes client requests for information in the form of HTTP GET commands, returning the information as a speakable string. Because of its lack of reliance on other libraries, this section is platform independent, taking any capable client, and running on any PHP server.

```
1  <?php
2
3  $InterroNouns = array("who", "what", "when", "where",
4                        "why", "how", "which"        );
5  $Copulas = array("is", "was", "are");
6  $Articles = array("the", "a", "an");
7  $Prepositions = array("of", "for");
8  $NoCapitalize = array("of", "for");
9
10 $InfoCommands = array("tell", "describe");
11 $ActionCommands = array("call", "send", "open");
12
13 //Classification ENUMs
14 abstract class WordType {
15         const InterrogativeNoun = 0;
16         const Copula = 1;
17         const Article = 2;
18         const Subject = 3;
19 }
```

```
20
21  abstract class SpecialInterroType {
22          const Location = 0;
23          const NOTA = 1;
24  }
25
26  abstract class SpecialWordType {
27          const Weather = 0;
28          const NoCapitalize = 1;
29          const NOTA = 2;
30  }
31
32  abstract class ClauseType {
33          const AskInfo = 0;
34          const Subject = 1;
35          const Unknown = 2;
36          const NOTA = 3;
37  }
38
39  abstract class ActionType {
40          const Description = 0;
41          const Command = 1;
42          const Definition = 2;
43          const Converse = 3;
44          const Unknown = 4;
45  }
46
47  abstract class ConceptType {
48          const Person = 0;
```

```
49          const Place = 1;
50          const Noun = 2;
51  }
52
53  //Utilities
54  function CapitalizeFirst($Word) {
55          global $NoCapitalize;
56          $Low = strtolower($Word);
57          for ($i = 0;
58               $i < count($NoCapitalize);
59               $i++) {
60                  if (strcasecmp($Low,
61                                 $NoCapitalize[$i]) == 0) {
62                          return $Low;
63                  }
64          }
65          $Upper = strtoupper($Word);
66          $F = substr($Upper, 0, 1);
67          $B = substr($Low, 1);
68          return $F . $B;
69  }
70
71  function ClassifyWord($Word) {
72          global $InterroNouns,$Copulas, $Articles;
73          for ($i = 0;
74               $i < count($InterroNouns);
75               $i++) {
76                  if (strcasecmp($Word,
77                                 $InterroNouns[$i]) == 0) {
```

```
78                          return WordType::InterrogativeNoun;
79                  }
80          }
81          for ($i = 0;
82               $i < count($Copulas);
83               $i++) {
84                  if (strcasecmp($Word, $Copulas[$i]) == 0) {
85                          return WordType::Copula;
86                  }
87          }
88          for ($i = 0;
89               $i < count($Articles);
90               $i++) {
91                  if (strcasecmp($Word, $Articles[$i]) == 0) {
92                          return WordType::Article;
93                  }
94          }
95          return WordType::Subject;
96  }
97
98  function RetrieveInfo($R) {
99          $ExtPat = "/^[^\.]+/";
100         $Query = $R;
101
102         $XMLe = simplexml_load_file("https://en.wikipedia.org/
103                                      w/api.php?action=query
104                                      &prop=extracts&format=xml
105                                      &exintro=&explaintext=
106                                      &exsectionformat=plain
```

```
107                                     &indexpageids=
108                                     &exportnowrap=&iwurl=
109                                     &titles=" . $Query .
110                                    "&redirects=");
111
112         $Str = strip_tags($XMLe->query->pages[0]->page
113                                                 ->extract);
114         $Ret = "";
115
116         //Delim Capture requires parenthetical notation
117         $Ret = preg_split("/([^A-Z\.][\.])\s[A-Z]/", $Str,
118                           -1, PREG_SPLIT_DELIM_CAPTURE);
119
120         if (count($Ret) > 1) {
121                 return $Ret[0] . $Ret[1];
122         }
123         else {
124                 return $Ret[0];
125         }
126 }
127
128 //Classes
129 class Word {
130         private $Type;
131         private $Value;
132         function __construct($I) {
133                 $this->Type = ClassifyWord($I);
134                 $this->Value = $I;
135         }
```

```
136         function getType() {
137                 return $this->Type;
138         }
139         function getValue() {
140                 return $this->Value;
141         }
142         function capitalize() {
143                 $this->Value = CapitalizeFirst($this->Value);
144         }
145 }
146
147 class Clause {
148         private $Type;
149         private $Words;
150         function __construct($T) {
151                 $this->Type = $T;
152         }
153         function add($A) {
154                 $this->Words[] = $A;
155         }
156         function setType($T) {
157                 $this->Type = $T;
158         }
159         function getType() {
160                 return $this->Type;
161         }
162         function getWords() {
163                 return $this->Words;
164         }
```

```
165         function getSubject() {
166                 $S = "";
167                 for ($i = 0;
168                      $i < count($this->Words);
169                      $i++) {
170                         if ($this->Words[$i]->getType()
171                             == WordType::Subject) {
172                                 $S = $S . " " .
173                                      $this->Words[$i]
174                                          ->getValue();
175                         }
176                 }
177                 return substr($S, 1); //Remove Leading Space
178         }
179 }
180
181 function IdentifyClause($WordSet) {
182         $Len = count($WordSet);
183         switch($Len) {
184                 //AskClause?
185                 case 2:
186                         if ($WordSet[0]->getType() ==
187                             WordType::InterrogativeNoun) {
188                                 if ($WordSet[1]->getType() ==
189                                     WordType::Copula) {
190                                         return ClauseType
191                                             ::AskInfo;
192                                 }
193                                 else {
```

```
194                                         return ClauseType
195                                             ::NOTA;
196                                 }
197                         }
198                         else {
199                                 return ClauseType::NOTA;
200                         }
201                 break;
202
203                 default:
204                         return ClauseType::NOTA;
205                 break;
206         }
207 }
208
209 $Query = $_GET['r'];
210 $Source = explode(" ", $Query);
211
212 /*        £Classifications */
213 //Classify Words
214 for ($i = 0;
215      $i < count($Source);
216      $i++) {
217         $Classifications[$i] = new Word($Source[$i]);
218 }
219
220 //Identify Subject Clauses
221 $ClauseMode = ClauseType::NOTA;
222 /* £Clauses */
```

```
223 $CurrentPos = -1;
224 for ($i = 0;
225      $i < count($Classifications);
226      $i++) {
227         //Article starts new SubjectClause
228         if ($Classifications[$i]->getType() ==
229              WordType::Article) {
230                 $ClauseMode = ClauseType::Subject;
231                 $CurrentPos++;
232                 $Clauses[$CurrentPos] = new Clause(
233                                           ClauseType::Subject);
234                 $Clauses[$CurrentPos]->add(
235                                         $Classifications[$i]);
236         }
237         elseif ($Classifications[$i]->getType() ==
238                 WordType::Subject){
239                 //Wikipedia API Compatability
240                 $Classifications[$i]->capitalize();
241                 if ($ClauseMode == ClauseType::Subject) {
242                         //Add Subject to Existing
243                         //Subject Clause
244                         $Clauses[$CurrentPos]
245                        ->add($Classifications[$i]);
246                 }
247                 else { //Start new Subject Clause
248                         $ClauseMode = ClauseType::Subject;
249                         $CurrentPos++;
250
251                         $Clauses[$CurrentPos] = new Clause(
```

```
252                                                 ClauseType
253                                                ::Subject);
254
255                         $Clauses[$CurrentPos]
256                        ->add($Classifications[$i]);
257                 }
258         }
259         else {
260                 if ($ClauseMode == ClauseType::Subject) {
261                         //End Clause
262                         $ClauseMode = ClauseType::Unknown;
263                         $Clauses[$CurrentPos] = new Clause(
264                                                   ClauseType
265                                                  ::Unknown);
266                         $Clauses[$CurrentPos]
267                        ->add($Classifications[$i]);
268                 }
269                 elseif ($ClauseMode == ClauseType::Unknown) {
270                         //Continue Unknown Clause
271                         $Clauses[$CurrentPos]
272                        ->add($Classifications[$i]);
273                 }
274                 else {
275                         //Start Unknown Clause
276                         $ClauseMode = ClauseType::Unknown;
277                         $CurrentPos++;
278
279                         $Clauses[$CurrentPos] = new Clause(
280                                                   ClauseType
```

```
281                                                 ::Unknown);
282
283                         $Clauses[$CurrentPos]
284                        ->add($Classifications[$i]);
285                 }
286         }
287 }
288
289 //Identify Remaining Clauses, Starting with Longest Possible
290 /* £RequestClauses */
291 for ($i = 0;
292      $i < count($Clauses);
293      $i++) {
294         if ($Clauses[$i]->getType() == ClauseType::Unknown) {
295                 $Remain = true;
296                 while ($Remain) {
297                         $Clauses[$i]->setType(
298                         IdentifyClause($Clauses[$i]
299                                         ->getWords()));
300                         $RequestClauses[$i] = $Clauses[$i];
301                         $Remain = false;
302                 }
303         }
304         else {
305                 $RequestClauses[$i] = $Clauses[$i];
306         }
307 }
308
309 //Identify Action
```

```
310 $DesiredAction = ActionType::Unknown;
311 $Len = count($RequestClauses);
312 switch($Len) {
313         case 2:
314                 if ($RequestClauses[0]->getType()
315                     == ClauseType::AskInfo) {
316                         if ($RequestClauses[1]->getType()
317                             == ClauseType::Subject) {
318                                 $DesiredAction = ActionType
319                                                 ::Description;
320                         }
321                 }
322         break;
323 }
324
325 switch($DesiredAction) {
326         case ActionType::Command:
327                 echo $Answer =
328                      "I'm sorry, but I don't know how
329                       to do that.";
330         break;
331         case ActionType::Description:
332                 $Answer = RetrieveInfo(
333                           $RequestClauses[1]->getSubject());
334                 if (strcasecmp($Answer, "") != 0) {
335                         echo $Answer;
336                         break;
337                 }
338         default:
```

```
339                 echo $Answer =
340                       "I'm sorry, but I don't know what
341                        you mean.";
342 }
343
344 ?>
```

C.3.ii: Client-Side (Java)

The following shows the client-side code, written in Java. This code is specific to Android, due to its reliance on Google's APIs, and its implementation of the GUI.

```
1  package org.threepointlabs.sia;
2
3  import java.io.IOException;
4  import java.io.InputStream;
5  import java.io.UnsupportedEncodingException;
6  import java.net.URLEncoder;
7  import java.util.ArrayList;
8  import java.util.HashMap;
9
10 import org.apache.http.HttpEntity;
11 import org.apache.http.HttpResponse;
12 import org.apache.http.client.HttpClient;
13 import org.apache.http.client.methods.HttpGet;
14 import org.apache.http.impl.client.DefaultHttpClient;
15 import org.apache.http.protocol.BasicHttpContext;
16 import org.apache.http.protocol.HttpContext;
17
```

```
18 import android.os.AsyncTask;
19 import android.os.Bundle;
20 import android.app.Activity;
21 import android.content.Intent;
22 import android.speech.RecognizerIntent;
23 import android.speech.tts.TextToSpeech;
24 import android.speech.tts.OnUtteranceCompletedListener;
25 import android.speech.tts.UtteranceProgressListener;
26 import android.util.Log;
27 import android.view.LayoutInflater;
28 import android.view.Menu;
29 import android.view.View;
30 import android.view.ViewGroup;
31 import android.widget.TextView;
32
33 public class MainActivity extends Activity {
34         protected TextToSpeech TTSEngine;
35         protected String ClientQuery;
36         protected String ServerResponse;
37         protected String LocalServer =
38                         "http://192.168.1.222/";
39         protected String RemoteServer =
40                         "http://24.127.202.190/";
41         protected String Prefix = "app/autodecide?r=";
42         protected View ActivateButton;
43         protected View ProgressIndicator;
44         protected ViewGroup LogGroup;
45         protected int NumLines;
46
```

```
47          @Override
48          protected void onCreate(Bundle savedInstanceState) {
49                  super.onCreate(savedInstanceState);
50                  setContentView(R.layout.activity_main);
51                  ActivateButton = findViewById(
52                                  R.id.button_activate);
53                  ProgressIndicator = findViewById(
54                                      R.id
55                                       .progressbar_indicator);
56                  LogGroup = (ViewGroup) findViewById(
57                                        R.id.linearlayout_log);
58                  NumLines = 0;
59          }
60
61          public void onButtonPress(View view) {
62                  activateSpeech(42);
63          }
64
65          public void activateSpeech(int requestCode) {
66                  //Prepare TTS first, takes a bit
67                  TTSEngine = new TextToSpeech(this, null);
68                  showIndicator();
69                  //Create and Send Speech Recognition Intent
70                  Intent RI = new Intent(
71                                  RecognizerIntent
72                                 .ACTION_RECOGNIZE_SPEECH);
73                  RI.putExtra(RecognizerIntent
74                             .EXTRA_LANGUAGE_MODEL,
75                              RecognizerIntent
```

```
76                      .LANGUAGE_MODEL_FREE_FORM);
77              RI.putExtra(RecognizerIntent.EXTRA_PROMPT,
78                      "Please Speak Now");
79              startActivityForResult(RI, requestCode);
80      }
81
82      protected void onActivityResult(int requestCode,
83                                      int resultCode,
84                                      Intent data) {
85              super.onActivityResult(requestCode,
86                                     resultCode,
87                                     data);
88              if ((resultCode == RESULT_OK) &&
89                  (data != null)) {
90                      ArrayList<String> Results =
91                      data.getStringArrayListExtra(
92                           RecognizerIntent
93                          .EXTRA_RESULTS);
94                      ClientQuery = Results.get(0);
95                      addRow(ClientQuery);
96                      Log.d("", ClientQuery);
97      }
98      ProcessSpeechTask PST = new ProcessSpeechTask();
99      PST.execute();
100     }
101
102     protected void hideIndicator() {
103             ProgressIndicator.setVisibility(
104                             View.GONE);
```

```
105
106             ActivateButton.setVisibility(
107                             View.VISIBLE);
108         }
109
110         protected void showIndicator() {
111                 ActivateButton.setVisibility(
112                             View.GONE);
113                 ProgressIndicator.setVisibility(
114                                 View.VISIBLE);
115         }
116
117         protected void addRow(String Value) {
118                 ViewGroup newView = (ViewGroup) LayoutInflater
119                                     .from(this).inflate(R
120                                     .layout.log_item,
121                                      LogGroup, false);
122                 ((TextView) newView.findViewById(R
123                                    .id.logtextvalue))
124                                    .setText(Value);
125                 LogGroup.addView(newView, NumLines);
126                 NumLines++;
127         }
128
129         private class ProcessSpeechTask
130               extends AsyncTask<Void, Void, String> {
131                 protected String doInBackground(Void...
132                                                 params) {
133                         HttpClient LocalClient
```

```
134                             = new DefaultHttpClient();
135                 HttpContext LocalContext
136                              = new BasicHttpContext();
137             try {
138                 ClientQuery = URLEncoder
139                             .encode(ClientQuery,
140                             "UTF-8");
141             }
142         catch(UnsupportedEncodingException e) {
143                 e.printStackTrace();
144         }
145         HttpGet Get = new HttpGet(RemoteServer
146                                 + Prefix
147                                 + ClientQuery);
148         try {
149                 HttpResponse Response = LocalClient
150                                       .execute(Get,
151                                        LocalContext);
152
153                 HttpEntity Entity = Response
154                                   .getEntity();
155
156                 ServerResponse = getASCIIContent(
157                                  Entity);
158
159                 Log.d("", ServerResponse);
160
161                 HashMap<String, String> TTSParam =
162             new HashMap<String, String>();
```

```
163                     //Utterance ID needed for Callback
164                     TTSParam.put(TextToSpeech.Engine
165                                 .KEY_PARAM_UTTERANCE_ID,
166                                 "stringId");
167                     TTSEngine.speak(ServerResponse,
168                   TextToSpeech.QUEUE_ADD, TTSParam);
169
170                     runOnUiThread(new Runnable() {
171                             public void run() {
172                                     addRow(
173                                     ServerResponse);
174                             }
175                     });
176                 }
177             catch (Exception e) {
178                     return e.getLocalizedMessage();
179             }
180             //Release TTS when done
181             TTSEngine.setOnUtteranceProgressListener(
182                   new TTSAutoShutdown());
183             return null;
184     }
185
186     protected String getASCIIContent(HttpEntity Entity)
187               throws IllegalStateException, IOException {
188             InputStream IS = Entity.getContent();
189             StringBuffer SB = new StringBuffer();
190             int N = 0;
191             do {
```

```
192                         byte[] B = new byte[4096];
193                         N = IS.read(B);
194                         if (N > 0) {
195                                 SB.append(new String(B, 0,
196                                                         N));
197                         }
198                 } while (N > 0);
199                 return SB.toString();
200         }
201     }
202         protected class TTSAutoShutdown
203                 extends UtteranceProgressListener {
204
205                 @Override
206                 public void onDone(String arg0) {
207                         TTSEngine.shutdown();
208                         runOnUiThread(new Runnable() {
209                                 public void run() {
210                                         hideIndicator();
211                                  }
212                         });
213                 }
214
215                 @Override
216                 public void onError(String arg0) {
217
218                 }
219
220                 @Override
```

```
221                 public void onStart(String arg0) {
222
223                 }
224         }
225 }
226
```

Section D: Solution Evaluation

D.1: Criteria Met

The developed solution only meets most of the criteria originally presented earlier. Variation and comprehension of prior context were found to be difficult to implement, as the latter requires rewriting a significant portion of the english linguistic model, and the former requires a model of its own. Changing or making a linguistic model takes a lot of time, being the core, as well as the most complicated part of the project as a whole.

SIA does a decent job of maintaining the cognitive process, using an encyclopedic manner of speech (modeled after Wikipedia).

D.2: Annotated Test Output

D.2.i: Operation

Figure 6 shows the main screen before any input or interaction is made, and shows how the layout uses a similar design to existing agents, placing an activation button centered at the bottom of the screen underneath a transcript. Its important to place the activation button this way to make it usable with the thumb, without having the user shift his/her hand up, as well as only needing one hand for operation. The button's central placement makes the

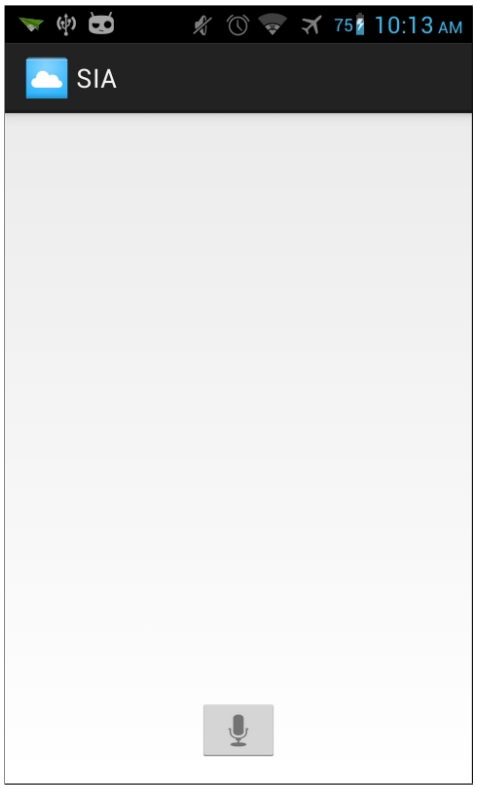

Figure 6: Main Screen

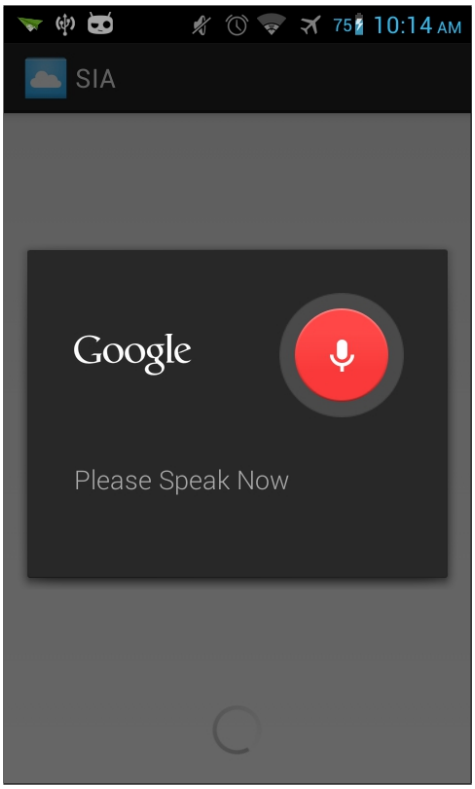

Figure 7: Voice Recognition Input

one-handed use ambidextrous. Figure 7 shows the voice recognition dialog that is shown when it is activated. The program piggybacks off of Google's Voice Recognition API, both because voice recognition is not the concept being developed, and because it is already well established.

Figure 8 shows how the query by the user is added to the transcript. Figure 9 shows how, in addition to being spoken, the response is also added to the the transcript. Both Figures 8 and 9 show how dividers are added to the transcript to help denote the different speech transactions. Figure 10 shows several additional examples. Both Figures 8 and 10 shows how the activation button is converted to a progress indicator when processing input or using the text-to-speech engine to audibly output the response.

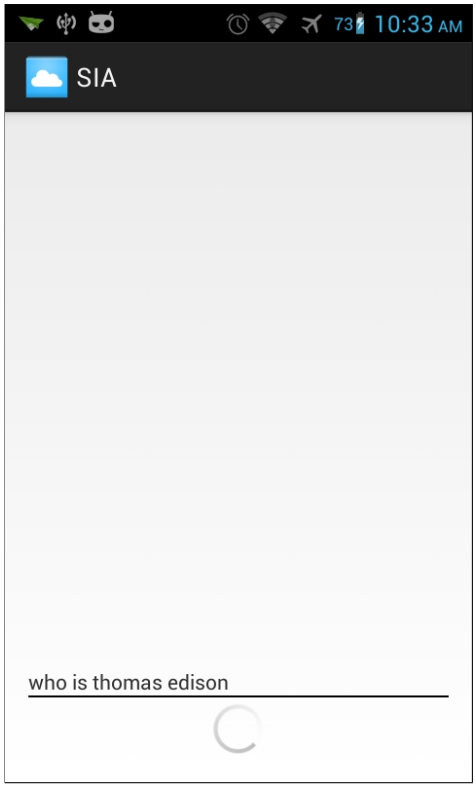

Figure 8: Query

Figure 9: Response

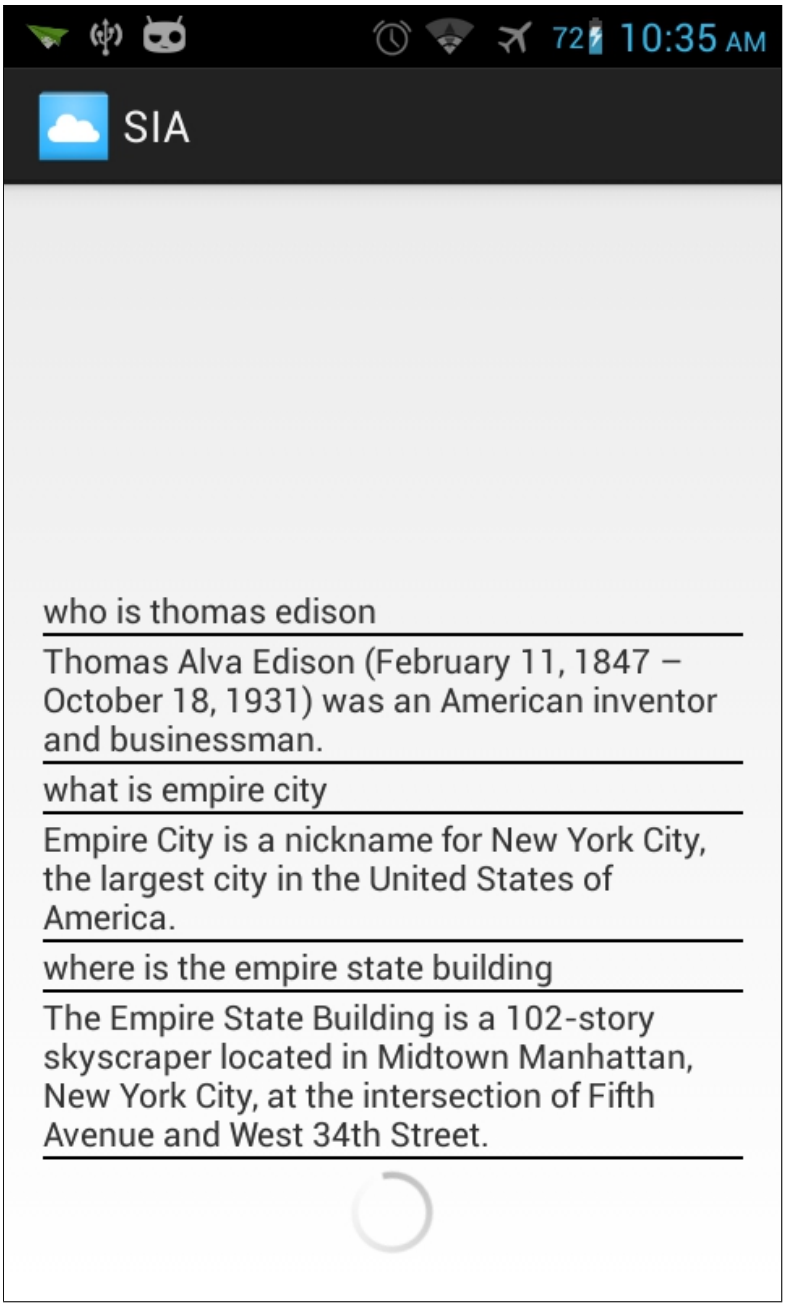

Figure 10: Additional Examples

D.2.ii: Error Handling

Figures 11 and 12 show how the program treats anything it cannot recognize through its defined english grammar model, whether it be due to a sentence that does not make any sense, or because it cannot infer what the user wants due to only including the desired subject.

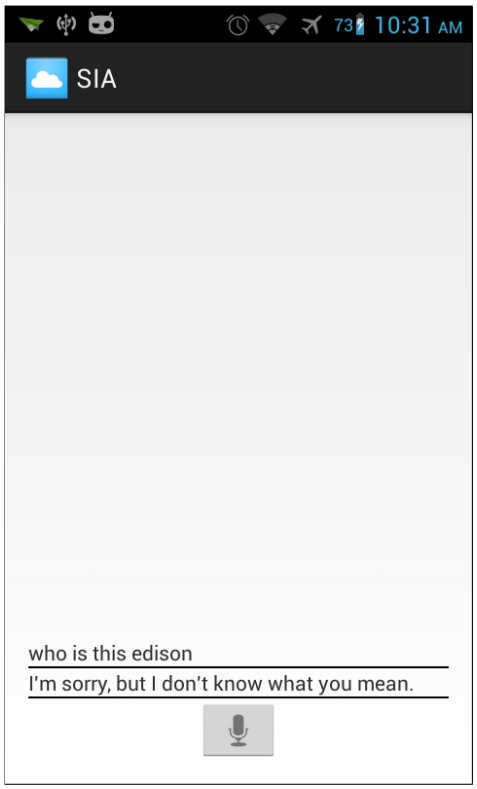

Figure 11: Improper Grammar

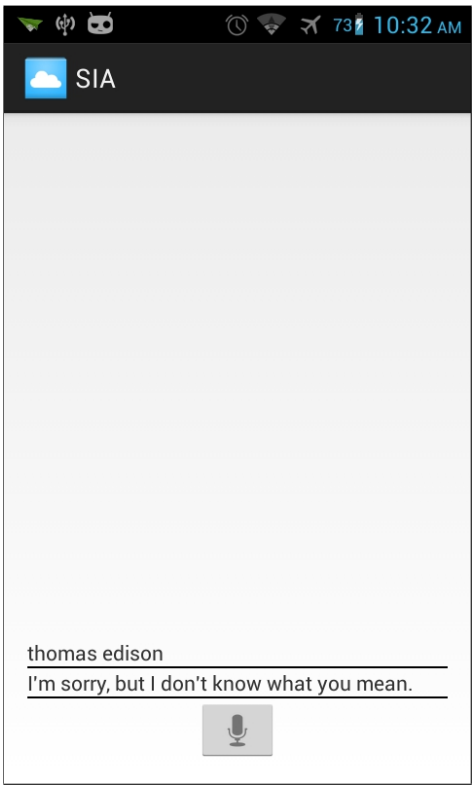

Figure 12: Not a Sentence

One of the major limitations of this implementation is its reliance on Google's APIs, making the program's input only as accurate as the Voice Recognition API in use. Figure 13 shows an incorrect recognition of the sentence "Who is Thomas Edison?", interpreting it instead as "food is on this has." Conveniently, Google's Voice Recognition API provides some as-

sistance with error handling, by re-prompting the user for input if none is detected, as shown in Figure 14.

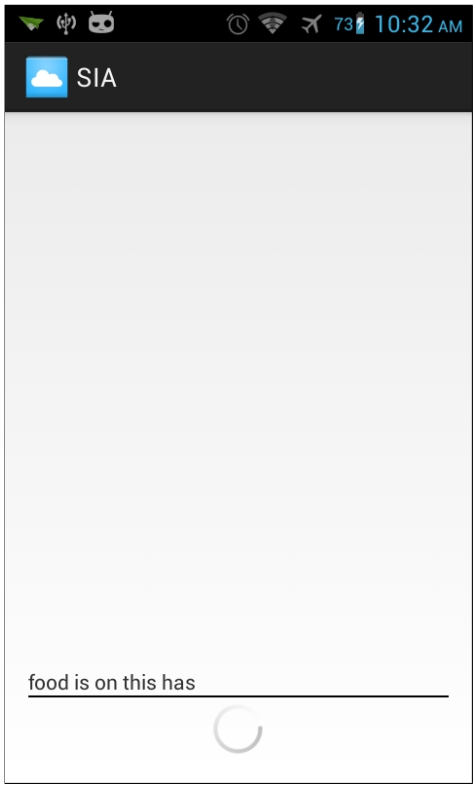

Figure 13: Incorrect Recognition

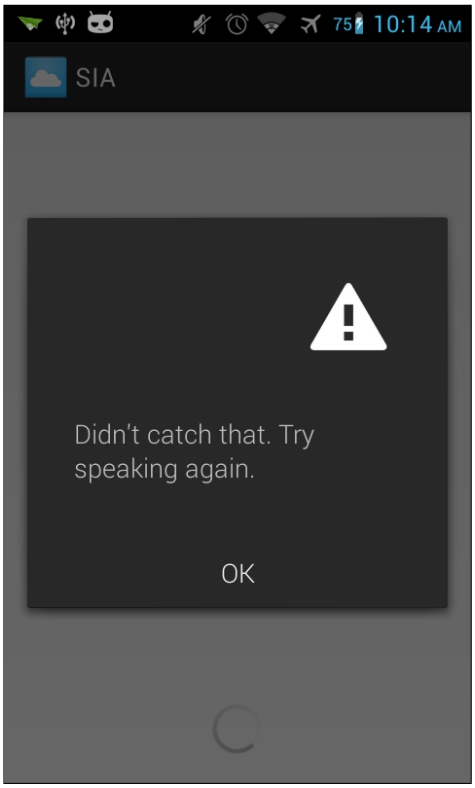

Figure 14: No Voice Input

D.3: Mastery Aspects

D.3.i: Advanced Mastery

- Parsing data stream - Java / PHP
- Inheritance - Java
- Encapsulation - Java / PHP
- Composition (hierarchical data structure) - PHP
- Abstract Data Types - PHP
- Use of Additional Libraries - Java

D.3.ii: Standard Mastery

- Arrays - Java / PHP
- User-defined Objects - PHP
- Simple Selection (if / else) - Java / PHP
- Complex Selection (nested if / multiple conditions / switch) - PHP
- Loops - Java / PHP
- Nested Loops - PHP
- Searching - PHP
- Use of Flags - Java / PHP

D.4: Conclusion

While the project did not meet all of the originally stated criteria, holistically it is a good implementation of the bigger picture, simplifying english's grammar model down to a specific set of words, while using pattern recognition to treat anything else as a subject, to make an agent that operates similarly to what one would expect from an AI. The grammar model created is far from complete, but handles one of the major expected abilities, being able to respond to most (general) queries of information.

Design-wise, the GUI could be used to differentiate between the user's input and the program's response, as both are currently formatted the same way, due to the single stack implementation. The original program design remains appropriate, leaving room for expansion through modification of the grammar model. One of the difficulties that will need to be addressed later on is porting to iOS. While Android does have a majority in the mobile OS market, it only sits at slightly over half, leaving nearly 40 percent unaccounted for. A port to iOS would expand coverage to between 80 and 90 percent. Porting the most complex part, the grammar model, is instantaneous because of its abstraction away from the client. However, due to Google's APIs (Voice Recognition and Text-to-Speech) being unavailable on Apple's iOS, and different UI design standards, the client will more or less have to be rewritten from the bottom up.