Smart Conversational User Interface for recommending Cultural Heritage Points of Interest

(Discussion Paper)

Flora Amato[1], Francesco Moscato[2], Vincenzo Moscato[3] and Giancarlo Sperlì[4]

[1]DIETI - University of Napoli Federico II, Naples, Italy
[2]DIEM. University of Salerno, Fisciano (SA), Italy
[4]DIETI - University of Napoli Federico II, via Claudio 21 - Naples, Italy
[4]DIETI - University of Napoli Federico II, via Claudio 21 - Naples, Italy

Abstract

Researchers and companies are making great efforts to create new ways of interaction with a increasing number and types of electronic devices, with particular attention to the conversational interface, whether spoken or written. The cultural heritage domain can bring many benefits from the great effort in this field, as a smart conversational system can provide both room/hotel/apartment information and information related to the area, natural or cultural points of interest, tourist routes, social events, history etc. The paper's main goal is to present a quick and effective service to the tourists visiting a city for the first time. Learning the cultural preferences of the users can enhance customer satisfaction, since a smart system can propose them customised tours in order to reach point of interests according to the expressed preferences. The interaction with the user is simplified by a conversational interface (chatbot).

1. Introduction

The Chatbot4Heritage (CB4H) project was born with the aim of integrating and centralising, on a single platform, information related to the tourism sector in Campania, generally from heterogeneous sources. In fact, the information a tourist may need during his or her stay ranges from details on local cultural sites, to upcoming cultural and entertainment events, the location of various types of services (e.g. in the catering sector), updates on the status of mobility, making it necessary to carry out numerous searches on different websites and worsening the overall user experience.

The project aims, through a service-oriented architecture, to extrapolate through a web crawler the information coming from multiple event aggregator websites, transform it into a common format and, through the mediation of an Enterprise Service Bus, store it in special databases. This information can then be made available to client applications, including a Chatbot[1, 2, 3] with which any users will be able to interact easily in natural language.

Researchers and companies are making great efforts to develop new ways of communicating with a increasing number and types of electronic devices, with particular attention to the conver-

flora.amato@unina.it (F. Amato); fmoscato@unisa.it (F. Moscato); vmoscato@unina.it (V. Moscato); giancarlo.sperli@unina.it (. G. Sperlì)

sational interface, whether spoken or in writing. The cultural heritage domain can bring many benefits from the great effort in this field, as an intelligent conversational framework can include both room/hotel/apartment information[4] and area specific information, natural or cultural points of interest, tourist routes, social events, history etc. The main aim of the paper is to provide the tourists visiting a city for the first time with a fast and efficient service.

The system can suggest to the users most appreciated cultural sites, by analysing opinion taken from social networks[5], and specifically taking in consideration cultural heritage communities[6] or the most authoritative persons[7] in the field of cultural heritage domain. Therefore, understanding the users' cultural preferences can improve customer loyalty, as a smart system can give them personalised tours to reach the point of interest according to the preferences expressed. A conversational interface[8] (chatbot) simplifies communications with the user.

Moreover, since of Covid-19 disease spreading, many cities started to improve their mobility services, like underground, by implementing a routing service for tourists[9, 10], that offers:

- Route planning by taking as input user's first and last stop.
- number of stops from starting stop to last one.
- Number of line changes during the route.
- Detection of the nearest stop to a given tourist's attraction.

TravelBot[8] provides the aforementioned basic functions and other services like:

- Informing the user about any faults on the lines.
- Showing the metro's map.
- Informing and guiding the user to the purchase of tickets.
- Showing the order of the stops for the line selected.
- Informing the user about the working timetables of metro's services.

2. System Architecture

In order to retrieve information about locations, events or other relevant information, we have to chose a set of sources considered reliable. Then, the obtained data-set must undergo a process of processing and cleaning, removing formatting errors and non machine readable elements, so that the information contained in it can be inserted in a natural language sentence[11].

We adopted a Service Oriented Architecture because of its intrinsic modularity. In this type of architecture, in fact, each module is developed independently and can be connected and removed from the main system without compromising the others. If, conceptually, the chatbot is considered a front end module, then it is possible to delegate data extraction and cleaning operations to other back end modules, using the Service Bus as an intermediary. The chatbot, at this point, can use the received data immediately, without having to process it further at runtime.

Some example components were chosen and described in Figure 1: a scraper, whose implementation in Python is described here through the Scrapy framework, for the Napolitoday.it website; an external API, that of the Yelp website; a MySQL database, which in this field is identified as Data Service; a web application in Django for direct insertion.

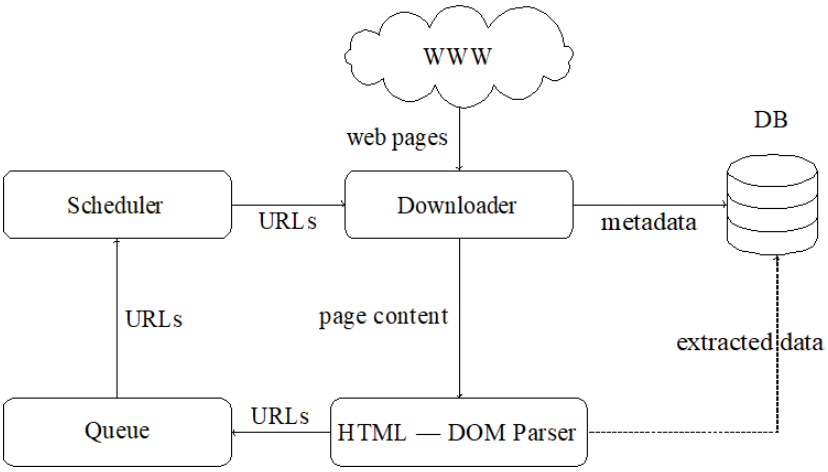

Figure 1: High level architecture of a bot. Data extraction occurs only in scraping bots.

For the implementation of the Service Bus, the choice fell on the open-source framework WSO2. WSO2 Enterprise Integrator has a graphical drag-and-drop interface for the configuration of the message flows between one component and another. One of the main features of a Service Oriented Architecture is the possibility to configure the interactions between components, instead of defining them entirely by code.

The application in Django, finally, was born as an interface for the management of CRUD (Crud-Retrieve-Update-Delete) operations by an employee or operator of a cultural entity. However, due to restrictions imposed by the framework itself, it was decided to use it for insertion only. In the section dedicated to future developments, this issue will also be dealt with[12].

3. Knowledge Base population

Information extracted from web pages sources are processed and stored in the system' Knowledge Base. The Knowledge Base was implemented in Prolog language, taking advantage of its ease of use. The code will be divided into:

- Facts: they report certain data known by the machine.
- Rules: they manipulate the facts to obtain data useful to the end user.

The "machine learning" process is used to obtain useful information for the user (e.g.) and this will help the programmer who does not need to provide too many facts to the machine.

3.1. Facts

The machine knows two types of facts:

1. Connections (conn), composed by 3 arguments:
   a) First stop;

b) Second stop;
c) Median time between that two stops, it also serves to identify which line is it.

It establishes that there is a connection between the two stops and how long the subway takes to travel from one to another.

```
            conn(battistini, cornella, 2).

```

2. Locations (locate), composed by 2 arguments:
   a) List of one or more tourist attraction
   b) Nearest stop to the attraction list.

   It links a list of tourist attraction to their nearest stop.

```
        locate([s_pietro,musei_vaticani],ottaviano).

```

3.2. Rules

The rules can enhance the inferential process that retrieve customised information according to user preferences. Further enhancement can be taken with the exploitation of semantic techniques[13]

Conn1 The machine can determine only a one way connection with his known fact "conn". To be able to establish a bidirectional connection between two stops, the machine uses this mathematical expression: $\exists conn(X,Y,C) if \exists conn(Y,X,C)$

The OR statement is needed to get rid off infinite recursive call that can be possible in the next set of rules.

Append The "append" rule, like his name suggest, will take care of appending one element to an existing list and it will give a new resulting list as an output.

- $[Head1|Tail1] = List$ where you want to append an element.
- List2 = Element or list of elements that you wish to append.
- $[Head1|TailResult] = Result$.

First part of the code, by using a simple fact, establishes the result of appending an element to an empty list. In the second part then we will use a rule that with a recursive call, taking advantage of backtracking, delete the head of [Head1|Tail1] and continues until the list is empty, after that it starts to rebuilt a new list starting from tail to head, using as a tail List2. This rule will come in handy when we will need to find out how many line change we have to do during end user's journey.

```
append([],X,X).
append([Head1|Tail1],List2,[Head1|TailResult]):-
    append(Tail1,List2,TailResult).
```

Change We use average time to go from one stop to the next, to check how many lines the tourist must change.

- T = Average time between two stops.
- Lc = List of changes, initially empty and used as a temporary.
- $[T|Ris] = A$ second list of changes, needed to make append rule to work and to get a result.

The rule will initially check if given T value is inside Lc by using: not(member(T,Lc)). After that if the result is true, append rule will add the given T value to [T|Ris]. Otherwise we use append with an empty list, so it will not add nothing to [T|Ris]. Result will be a list containing a maximum of three elements, that it will be then used to determine how many lines the tourist must change.

```
change(T,Lc,[T|Ris]):-
    not(member(T,Lc)),
    append([T],Lc,[T|Ris]).
change(T,Lc,[T|Ris]):-
    member(T,Lc),
    append([],Lc,[T|Ris]).
```

Len len works pretty much like append with a recursive call (backtracking) of itself and using a given fact: len([],0). goal is different, we need to evaluate the length of a given list, in our case that list will be Lc.

- $[\_HEAD|TAIL] = List$.
- N = Number of elements.

Decr This simple rule will decrement P by 1 and it will store the result in Q. decr will work with change and len to work on Lc, we will see that our main goal is to get the number of elements in Lc and decrement that number by one to get our result.

Connection connection is a rule that takes two stops as input and calculates the list of stops, line changes made and duration time of the journey. The algorithm takes care of solving the "travel salesman" problem.

Rule's arguments are:

- Fs = First stop.
- Ls = Last stop.
- Ms = Middle stop.
- Lc = List of changes.
- Pl = Prohibited list.
- T = Total time of journey.
- N1 = Number of changes.
- Result = Non decremented list of changes.
- L1c = Temporary list with last stops to do.

In the first part of code, connection will check if there is a direct connection between first stop (Fs) and last stop (Ls) by using rule conn1 and checking after if these two stops are member of Pl. Otherwise if this first part is false, the machine will search for a middle stop Ms and check if there is a direct connection to first stop (Fs). When this stop is found, the machine will check if both stops are not in the prohibited list using not(member) after that change rule will be called checking if there was a line change from one stop to another. Change rule as we saw makes a list made of average times taken only once (in this specific case), so the maximum length of this list will be three, decreasing this value by 1 will give us N1. Please note: Both Pl and Lc must be empty list while calling connection in a query, because when the tourist starts his journey he have not travelled to anywhere still. Also neither first stop and last stop can be prohibited stops. Connection is going to use a recursive call that will find a connection between Ms and Ls, using updated arguments value:

$$Fs \rightarrow s \qquad Pl \rightarrow [Fs, Pl] \qquad Lc \rightarrow Result \tag{1}$$

The recursive call has to be called until it find a direct connection between middle stop (Ms) (which is going to be updated everytime) and final stop (Fs). After the rule will find final route it will determine length of Result by using change and len, lastly to get N1 it uses decr (N,N1). Then connection will calculate T by adding each time between all the stops.

```
connection(Fs,Ls,Pl,Lc,[Fs,Ls],T,N1):-
    conn1(Fs,Ls,T),
    not(member(Fs,Pl)),
    not(member(Ls,Pl)),
    change(T,Lc,Ris),
    len(Ris,N),
    decr(N,N1).
connection(Fs,Ls,Pl,Lc,[Fs|L1c],T,M):-
    conn1(Fs,Ms,T1),
    not(member(Fs,Pl)),
    not(member(Ms,Pl)),
    change(T1,Lc,Result),
    connection(Ms,Ls,[Fs,Pl],Result,L1c,T2,M),
    T is T1+T2.
```

Visit Rule that calculate the path for the nearest stop, based on the selected attraction, it also shows the travel time, the number of changes and the stops list. Or, choose the final stop you want to reach and define the initial stop where the user is located, check if there are tourist attractions nearby. Compared to 'Connection' rule, there is an additional argument: the attraction that the user wants to visit (Att). The IM, as we already said, shows facts that associate a list of attractions with their nearest stop. Arguments:

- Att = Attraction.
- Fs = First stop.
- Ls = Last stop.
- Ms =Middle stop.
- Lc = List of changes.

- Pl = Prohibited list.
- T = Total time of travel.
- N1 = Number of changes.
- Result = List containing the number of changes not decreased.
- L1c =List that represent the remaining path.

In the first case the IM checks if there is already an attraction or a list of them X, in the Knowledge Base, that can be associated with the last stop (Ls). After the IM finds it, checks if the desired attraction belongs to the list X, if it's true the IM searches a connection between Fs and Ls. If, at the first try, the IM doesn't find a direct connection, then it recalls the OR in visit and it will repeat the initial step, but this time searching, In the facts, a connection between Fs and a middle stop Ms, with T1 as travel time, recalling the rule: conn1. For the remaining part of the code, can be checked the rule: connection.

```
visit(Att,Fs,Ls,Pl,Lc,[Fs,Ls],T,N1):-
    locate(X,Ls),
    member(Att,X),
    conn1(Fs,Ls,T),
    not(member(Fs,Pl)),
    not(member(Ls,Pl)),
    change(T,Lc,Ris),
    len(Ris,N),
    decr(N,N1).
visit(Att,Fs,Ls,Pl,Lc,[Fs|L1c],T,M):-
    locate(X,Ls),
    member(Att,X),
    conn1(Fs,Ms,T1),
    not(member(Fs,Pl)),
    not(member(Ms,Pl)),
    change(T1,Lc,Result),
    visit(Att,Ms,Ls,[Fs,Pl],Result,L1c,T2,M),
    T is T1+T2.
```

4. TravelBot

In this section we will show the main functions of the chatbot component and guidelines for correct execution. The main program uses two doc.json, these two document has been made with two different approach to test the chat bot:

1. 'doc-v1.json' had pattern made of extended question containing when/how/what/which pronouns;
2. 'doc.json' is the current document in use, and is made of keywords for each tag with no pronouns at all.

If you want to test the two different documents please make sure to delete model[8] and data files or uncomment line 46 and 123, and change the following constant to the corresponding file.

In Listing 1 we open the document 'doc.json' and download the content. With the Boolean cycle 'for' we are taking the words taken from the document and inserting them in the lists through the functions 'append' and 'extend'.

```
with open(DOC_FILE) as file: #Opens "doc.json" file and load with json libray
    .
  data = json.load(file)

try:
  #asfgsadf <----- uncomment this to force data parsing again
  with open("data.pickle", "rb") as f:
    words, labels, training, output = pickle.load(f)
except:
  #Empty lists
  words = [] #Words list contains all the words of our "patterns"
  labels = [] #Labels list containing all the "tag" words
  docs_x = [] #doc containing a list of all the patterns with its "tag" in
    the corresponding position (docs_y)
  docs_y = []

  #Start cycling through out data dictionary (doc.json)
  for intent in data["intents"]:
    for pattern in intent["patterns"]:
      #tokenize all the world in our document using nltk, after that we
    extend word list,

      wrds = nltk.word_tokenize(pattern)
      words.extend(wrds)
      docs_x.append(wrds)
      docs_y.append(intent["tag"])

    if intent["tag"] not in labels:
      labels.append(intent["tag"])
```

Listing 1: File open

In Listing 2 we make all the words lowercase, and remove the character "?". After that we will stem the words, in other words we find the root version of our words. This will help us to see the dimension of the whole vocabulary not counting on duplicates.

```
  words = [stemmer.stem(w.lower()) for w in words if w not in "?"]
  words = sorted(list(set(words))) #Remove duplicates and sort my list of
    words.

  labels = sorted(labels) #Sorts labels list.
```

Listing 2: Text Normalization

5. Conclusion

The paper main goal is to present a fast and effective service to the tourist, who is visiting for the first time a city, full of culture and connections. In this work we verified the feasibility of a centralised service-oriented system aiming to the population of a Knowledge Base to be integrated into a tourism chatbot. Further development involves the enhancement of natural language processing algorithms to perform a data extraction based on page content, using the bot only to automate navigation or to support data already extracted.

Acknowledgement. This paper has been produced with the financial support of the Project financed by Campania Region of Italy "Synergy-net: ricerca e Digital Solutions nella lotta alle patologie oncologiche" CUP B61C17000090007. "POR CAMPANIA FESR 2014/2020 – ASSE 1 – O.S. 1.2 Avviso "Manifestazione di Interesse per la Realizzazione di Technology Platform nell'Ambito della Lotta alle Patalogie Oncologiche"Dec. N 460 DEL 28/11/2018. CINI"

References

[1] E. Adamopoulou, L. Moussiades, An overview of chatbot technology, in: Proc. IFIP Int. Conf. on Artificial Intelligence Applications and Innovations, Springer, 2020, pp. 373–383.

[2] J. L. Z. Montenegro, C. A. da Costa, R. da Rosa Righi, Survey of conversational agents in health, Expert Systems with Applications 129 (2019) 56–67.

[3] J. Pereira, Ó. Díaz, Using health chatbots for behavior change: a mapping study, Journal of medical systems 43 (2019) 135.

[4] F. P. Putri, H. Meidia, D. Gunawan, Designing intelligent personalized chatbot for hotel services, in: Proc. the 2019 Int. Conf. on Algorithms, Computing and Artificial Intelligence, 2019, pp. 468–472.

[5] F. Amato, G. Cozzolino, F. Moscato, V. Moscato, A. Picariello, G. Sperli, Data mining in social network, in: Procs. Intelligent Interactive Multimedia Systems and Services, Springer, 2018, pp. 53–63.

[6] A. Castiglione, G. Cozzolino, V. Moscato, G. Sperlì, Analysis of community in social networks based on game theory, in: Proc. IEEE Int. Conf. on Dependable, Autonomic and Secure Computing, Int. Conf. on Pervasive Intelligence and Computing, Int. Conf. on Cloud and Big Data Computing, Int. Conf. on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech), IEEE, 2019, pp. 619–626.

[7] F. Amato, G. Cozzolino, G. Sperlì, A hypergraph data model for expert-finding in multimedia social networks, Information 10 (2019) 183.

[8] R. Canonico, G. Cozzolino, A. Ferraro, V. Moscato, A. Picariello, F. R. Sorrentino, G. Sperlì, A smart chatbot for specialist domains, in: Workshops of the Int. Conf. on Advanced Information Networking and Applications, Springer, 2020, pp. 1003–1010.

[9] V. La Gatta, V. Moscato, M. Postiglione, G. Sperli, An epidemiological neural network exploiting dynamic graph structured data applied to the covid-19 outbreak, IEEE Transactions on Big Data (2020).

[10] A. Galli, M. Gravina, V. Moscato, G. Sperli, et al., Deep learning for hdd health assessment: an application based on lstm, IEEE Transactions on Computers (2020).

[11] F. Amato, F. Moscato, V. Moscato, F. Pascale, A. Picariello, An agent-based approach for recommending cultural tours, Pattern Recognition Letters 131 (2020) 341–347.

[12] A. Amato, G. Cozzolino, C'meal! the chatbot for food information, in: Proc. Int. Conf. on Intelligent Networking and Collaborative Systems, Springer, 2020, pp. 238–244.

[13] G. Cozzolino, Using semantic tools to represent data extracted from mobile devices, in: Proc. IEEE Int. Conf. on Information Reuse and Integration (IRI), IEEE, 2018, pp. 530–536.