Masterarbeit

Travel Service Composition under Temporal Constraints

Javadan Munawar

Course of Study: INFOTECH

Examiner: Prof. Dr. Marco Aiello
Supervisor: Prof. Dr. Marco Aiello

Commenced: October 15, 2019
Completed: June 29, 2020
Abstract

Web service can be a software, application or a document available over the internet, the ever-rising use of these services have built immense trust for the general public, thus resulting in the need to make better, smart and efficient web services that can help cater the needs of the online community. A high number of services are on offer for a specific task makes it very competitive for the service providers to have an edge. When users start to make online requests for services it can result in rather complex queries which might not be solved by a single service and then service composition comes to play. There can be several methods, criterion and constraints which can be the basis of service composition. This Masters thesis was about composing services to solve queries that had some element of time, most of the times when a user makes a request to Virtual Personal Assistants like 'SIRI', these requests are simple and can be managed with ease using one service but sometimes if a user enters a complex request, it makes it a lot more demanding. Initially, the aim was to get the familiarity with the operations of how the assistants work, then get to know how the services are rendered for the user requests, then to get deep insight on how the services are composed and finally to set up an environment where some of the selected queries can be implemented. There were two options to have temporal reasoning based on the user queries, once after the query has been entered and secondly when the response from the service provider is received. After the research part, during the implementation phase, all the proposed and chosen queries were realised.
## Contents

1 Introduction 13  
1.1 Problem Statement and research questions 13  
1.2 Methodology 13  
1.3 Context of research and motivation 14  
1.4 Thesis structure and organization 14  

2 State of the art 17  

3 Background 23  
3.1 Personal assistants 23  
3.1.1 Daily life and family 24  
3.1.2 Context awareness of PAs 25  
3.1.3 Marketing PAs’ 26  
3.1.4 PAs and Learning 27  
3.1.5 PAs for task and time management 28  
3.1.6 Challenges and future of PAs 28  
3.2 Web Services 29  
3.2.1 Discovery of semantic web services 30  
3.2.2 Web services discovery with QoS 31  
3.2.3 Trust in web services 32  
3.2.4 RESTful versus Big web services 32  
3.3 Service composition 33  
3.4 Techniques for temporal reasoning 36  
3.4.1 Allen’s Temporal intervals 36  
3.4.2 Linear Temporal Logic 38  
3.4.3 Computational Tree Logic 38  

4 System design and execution flow 41  
4.1 Service composition for a trip 41  
4.2 Getting started with basic APIs 45  
4.3 Case study’s block diagram 48  
4.4 Sample queries 49  
4.4.1 Query: One way flight 49  
4.4.2 Query: Return flight 51  
4.4.3 Query: To find cheapest routes on specific dates 51  
4.4.4 Query: For enlisting all flights in a month 52  
4.4.5 Query: Enlisting all possible flights in next week 53  
4.4.6 Query: Possible flights on a specific day next week 55  
4.5 User Interface for a query 55
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Query: One way flight</td>
<td>59</td>
</tr>
<tr>
<td>5.2</td>
<td>Query: Return flight</td>
<td>60</td>
</tr>
<tr>
<td>5.3</td>
<td>Query: To find cheapest routes on specific dates</td>
<td>61</td>
</tr>
<tr>
<td>5.4</td>
<td>Query: For enlisting all flights in a month</td>
<td>62</td>
</tr>
<tr>
<td>5.5</td>
<td>Query: Enlisting all possible flights in next week</td>
<td>64</td>
</tr>
<tr>
<td>5.6</td>
<td>Query: Possible flights on a specific day next week</td>
<td>65</td>
</tr>
<tr>
<td>6.1</td>
<td>Conclusion</td>
<td>67</td>
</tr>
<tr>
<td>6.2</td>
<td>Future work</td>
<td>67</td>
</tr>
<tr>
<td>Bibliography</td>
<td></td>
<td>69</td>
</tr>
</tbody>
</table>
List of Figures

1.1 Personal assistants usage numbers [Ste17] . . . . . . . . . . . . . . . . . . . . . 14
1.2 Number of installed PAs [Bre19] [Sev19] . . . . . . . . . . . . . . . . . . . . . 14
2.1 Business model example (using finite state machine) [BCPT05] . . . . . . . . . 17
2.2 Temporal aware service requests and offers [MRDM05] . . . . . . . . . . . . . 18
2.3 Geographical and Temporally different WS requests [ZSLG14] . . . . . . . . . 19
2.4 Basic approach for WS composition [ZLH+14] . . . . . . . . . . . . . . . . . . 19
2.5 Basic approach for WS composition [ZLH+14] . . . . . . . . . . . . . . . . . . 20
2.6 LTL approach for WS composition [HZ10] . . . . . . . . . . . . . . . . . . . . . 20
3.1 PA usage example . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24
3.2 PA with personalization [WHM+02] . . . . . . . . . . . . . . . . . . . . . . . . 25
3.3 Usage of PAs in US survey [Bre19] . . . . . . . . . . . . . . . . . . . . . . . . . 26
3.4 Usage tree for selecting meeting location [MCF+99] . . . . . . . . . . . . . . . 27
3.5 Interaction between client and service provider [Kre01] . . . . . . . . . . . . . 29
3.6 Web service architecture including roles and operations [Kre01] . . . . . . . . 30
3.7 New model for Web Service discovery [Ran03] . . . . . . . . . . . . . . . . . . . 31
3.8 Approaches for trust based service selection [APC10], [Alj15] . . . . . . . . . 32
3.9 Application integration styles [PZL08] . . . . . . . . . . . . . . . . . . . . . . . . 33
3.10 WS composition infrastructure [PBM02] . . . . . . . . . . . . . . . . . . . . . . 34
3.11 WS composition infrastructure based on [OYP03a] . . . . . . . . . . . . . . . . 35
3.12 Basic Allen interval relations [KJJ03] . . . . . . . . . . . . . . . . . . . . . . . 36
3.13 Constraint graph and a possible solution [Dec03] . . . . . . . . . . . . . . . . 37
3.14 LTL modal diagram example [Art11] . . . . . . . . . . . . . . . . . . . . . . . . 38
3.15 Branched modal diagram example [Art11] . . . . . . . . . . . . . . . . . . . . . 39
3.16 CTL example [Fou05] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39
4.1 Logistic management system diagram . . . . . . . . . . . . . . . . . . . . . . . 41
4.2 Hotel management system diagram . . . . . . . . . . . . . . . . . . . . . . . . 42
4.3 Temporal aspect involving hotel booking . . . . . . . . . . . . . . . . . . . . . 43
4.4 Travel management system diagram . . . . . . . . . . . . . . . . . . . . . . . . 43
4.5 Possible temporal reasoner flow diagram . . . . . . . . . . . . . . . . . . . . . 44
4.6 Complete block diagram for the thesis . . . . . . . . . . . . . . . . . . . . . . . 48
4.7 Display of sample JSON file on left and the converted table from this on right . 53
4.8 Display of the GUI for one of the queries . . . . . . . . . . . . . . . . . . . . . . 56
4.9 GUI with example data . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 57
4.10 Possible temporal reasoner flow diagram . . . . . . . . . . . . . . . . . . . . . 57
5.1 Example input for query 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 59
5.2 Example output in json format . . . . . . . . . . . . . . . . . . . . . . . . . . . 59
List of Tables

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Highlighting APIs and Web services [Hug18]</td>
<td>29</td>
</tr>
<tr>
<td>3.2</td>
<td>SC models [ACKM04]</td>
<td>33</td>
</tr>
<tr>
<td>3.3</td>
<td>Intervals and their possible relations</td>
<td>37</td>
</tr>
<tr>
<td>3.4</td>
<td>Operators used in LTL [Art11] [Wik20b]</td>
<td>38</td>
</tr>
</tbody>
</table>
List of Listings

4.1 The code shows authentication requirements. ........................................ 45
4.2 The code shows the libraries that are to be used .................................. 45
4.3 The code shows the basic implementation of an API in Python ............... 46
4.4 The code shows the request.get command in usage ............................. 46
4.5 How to retrieve relevant information from the received JSON ................ 47
4.6 The code shows the JSON response .................................................. 47
4.7 The required information after using listing 4.6 ................................. 47
4.8 The required libraries in Python ...................................................... 49
4.9 The function made for custom printing ............................................. 49
4.10 The response with out custom printing ............................................. 50
4.11 Relatively understandable form ...................................................... 50
4.12 The code that allows input of date in specific format ............................ 51
4.13 The code that allows URL encoding ............................................... 51
4.14 The inputs taken for origin and destination cities ............................... 52
4.15 This code is used for selection of right month based on user request ....... 52
4.16 This code is used for selection of right week based on user request ....... 54
4.17 This code is used for selection of right week based on user request ....... 54
4.18 This code is used for selection of right week and day based on user request 55
1 Introduction

1.1 Problem Statement and research questions

Virtual PAs can perform a lot of small tasks based on the request of the user but performing tasks that involve a lot of services to be combined to meet the desired outcome is rare and service composition based on the temporal aspect of the user query is even rarer. This thesis will look into ways these services can be combined to perform time-related queries. The following points are the research questions which will be the sole focus of the thesis.

- How personal assistants, web service composition and temporal reasoning will work?
- Find out which APIs can be used to model an application based on the research?
- Devise and see how different queries can be set up in a way to have some sort or temporal aspect involved?
- Finding an appropriate coding environment that can be used to implement the API composition and how will the devised queries be implemented in this application?

1.2 Methodology

The main aim of this thesis will be to figure out how services can be composed or how an application can provide the requester with the desired results based on the queries that have some sort of temporal aspect to them. The methodology here would be to look for work that has been done in the past to give us the head-start and then look for how we can make some prototype application to solve basic queries with temporal constraints.

- We will start with looking for scenarios like travel, logistics, hotel booking and several others that can have few queries that can have time related constraints
- We will preempt few queries that a user can have and involves time as well
- We will look for APIs that can be used to meet the selected queries
- making an application or a wrapper to be specific that can make use of all the APIs and retrieve required information
- At the end to run some use cases and see how the system works.
1 Introduction

1.3 Context of research and motivation

Nowadays Personal Assistants (PAs) like Siri (Apple), Viv/Bixby (Samsung), Alexa (Amazon) and many others rely on online services provided by the third party to meet the demands or requests of the user. Sometimes the user request for example “what’s the maximum temperature today?” can be satisfied with the help of only one service. Most of the times a request can not be fulfilled with just one service, but it needs multiple services to be combined together. In this master thesis, the main aim is to investigate techniques that can smartly compose services and further some implementation can be done to show prototypes connecting web services to user requests.

The use case selected for this thesis is service composition for helping the user choose travel options and some examples(user requests) are shortlisted for implementing some prototypes. A software wrapper is to be made that can invoke multiple web services using the researched technique.

1.4 Thesis structure and organization

This thesis is divided into few important chapters and these chapters highlight the pivotal phases from research to implementation.
Chapter 1 (Introduction)
The first chapter was the introduction part, containing motivation and problem statement.

Chapter 2 (State of the art)
The second chapter is related to the state of the art, that is, the work done closely related to the topic of this thesis, which is adding temporal aspect for service invocation or searching for services based on temporal reasoning.

Chapter 3 (Background)
The third chapter goes into detail about all the important aspects of this thesis, like, explanation of what personal assistants are? and how they work? how important can the services be? that are provided over the web these days, how this can lead to service composition and finally which different techniques can be used for temporal reasoning for choosing web services.

Chapter 4 (System design and execution flow)
The fourth chapter provides an insight on all the decision making that was done when choosing a niche to work with and later part describes how this was done.

Chapter 5 (Implementation of use cases)
This chapter will contain all sample implementations of the prototype of the web application.

Chapter 6 (Conclusion and future work)
The fifth chapter would briefly describe what has been done in a nutshell and then some future work as it is mostly an implementation based project.
2 State of the art

In this chapter, we will look into some of the research work previously done which is closely related to our topic that is web service (WS) composition (WSC) using some sort of temporal aspect (related to time) to execute the user-desired functionality.

As web services are offering application services over the web to allow interaction between clients and provider, for the fulfillment of user requests thus leading to the need of defining time constraints, one such example is mentioned in [BCPT05], where temporal aspects of web services are modelled. This can be explained using an example from the real world, One of the most used WS nowadays is e-commerce where shopping can be done from the comfort of your house and now more than ever because of global lockdown due to the pandemic.

![Figure 2.1: Business model example (using finite state machine) [BCPT05]](image)

In the Figure 2.1 the dotted line arrows are for the implicit actions and the complete line arrows are for explicit actions. Here all the states represent the different phases one has to go through or might go through when shopping online, the switch from one state to the other is made possible using messages from the buyer to the provider (+) or in the other direction (-). Most of the transmissions happen explicitly, that means on the accord of the user (through messages or user responses online) but some transactions might happen implicitly and almost all of these are due to timed deadlines. Many WSs provide a certain feature or functionality for a certain period of time and after that, it might expire, so this is where the temporal aspect explained in [BCPT05] comes into play. As it can be seen from the state diagram that if a customer logs in to the WS and starts to scroll through all the items on sale and might add items to the cart for buying later and continue to window shop and after that might look to get a total quote for the order, now the implicit action can be seen...
as the quoted items can only stay in the cart for three days (4320 minutes) and after that, it will expire automatically. For example this series of state transitions (State name, time in minutes) are acceptable "(login(+),0) (searchGoods(+),1) (addToCart(+),2) (quoteRequest(+),3) (cancel(+),4) and this (login(+),0) (searchGoods(+),1) (addToCart(+),2) (quoteRequest(+),3) (cancel(+),4330) is not" [BCPT05].

Web service procurement (WSP) can be described as the automatic search and selection of WSs based on the user requirements, this selection is made on the basis of Quality of Service (QoS) on offer from the provider side, In [MRDM05] an approach to WSP that has the element related to time, is proposed, for example, a WS that offers some functionality that is used by office workers around the world, thus it should have the peak Uptime (availability) from 0900 to 1700 hours and the five working days of the week to provide the best service possible. Prior to this negligible amount of work was done to adhere to this kind of temporal aspect of provided services [MRDM05].

![Figure 2.2: Temporal aware service requests and offers [MRDM05]](image)

The temporal aspect of a WS is important when the demand for that particular service is 99% or very high during some time of the day and less otherwise. Even though a lot of complexity can be added to this example as working hours in the one-time zone will be different to the working hours of the others but to keep things simple, for the purpose of explanation, a basic example is taken in Figure 2.2. In Figure 2.2 both $\delta$ are the demands of the user and $\omega$ are the offers, demands are made by the user and offers are the best suitable responses the service provider can make, the first demand requests a slot for the time period 0800 to 1400 from any slot after the second slot and the second demand requests two different slots with two different intervals and the left-most graph depicts the possible response from the service provider that it can assign these two available slots to the user to fulfil its requirements [MRDM05].

As seen the usage of WSs is on an exponential rise thus resulting in a challenge to deliver services with sort of foreknown credibility (Quality of Service (QoS)). When a user looks for a certain online query to be solved, many WSs are shortlisted for the specific task by looking through the service registry and one with the highest QoS is invoked but the problem discussed in [ZSLG14] details that actual QoS value is quite different from the mentioned value in the registry. As mentioned it happens due to two reasons, firstly WS performance depends on the moment they are invoked (resulting in different workload, amount of active users and current network environment) and secondly the user’s location matters as one service can be requested for invocation from different
parts of the world resulting in a different time as shown in Figure 2.2. In this example, if users from London, Beijing, Los Angeles and New York request for the service at different times resulting in the need to predict the QoS value depending on the temporal aspect as well [ZSLG14].

Figure 2.3: Geographical and Temporally different WS requests [ZSLG14]

WSs provide platforms to develop applications by allowing interactions between respective entities to get the desired functionality. Currently WS description, discovery and invocation is a possibility but [Sin03] describes WSs perform at the highest level when composed in a novel way but currently the composition is limited to hard-coded compositions thus proposing to compose using realistic scenarios. WS composition can be divided into two categories using temporal aspect, firstly long-lived WSs, for example when a user, who bought something online and upon receiving is not satisfied by the delivered quality and now wants a refund, secondly the short-lived services like checking the user’s entered address that if the provider delivers in this region or not [Sin03].

As WS composition allows to configure multiple WSs together with an aim to reach ultimate user goal and QoS is important to find relevant WSs. In [ZLH+14], it is proposed to compose services using the fore mentioned novel approach which works by converting the already existing QoS aware task to a planning problem with added numerical and temporal aspect. As the number of WSs are increasing every given minute resulting in a lot of services that provide very similar functionality and thus ever more need to have better QoS management so that the user requests can be met efficiently, when talking about WS composition the challenging thing is to manage QoS constraints in specific regions and user queries in a way to develop composite services that meet all the set requirements [ZLH+14].

Figure 2.4: Basic approach for WS composition [ZLH+14]
In Figure 2.4 it can be seen as a simple example, the user wants to fulfil a request and there are eight sets of tasks to be completed before the query is properly serviced and every single task like T1 and T2 have many options that mean many services are offering to meet the requirements as shown in Figure 2.5, the main basic WS composition is to select one of the WS providers from each circle of the task and reach the endpoint [ZLH+14].

As the human dependency on WSs has increased in the last few years has resulted in an increased number of WSs responding to user problems thus making it even more difficult to choose an appropriate service at any particular occasion. To deal with this issue, like many other research works mentioned above and in [HZ10] another solution is proposed to automatically compose WSs keeping Linear Temporal Logic (LTL) in consideration [HZ10].

The service, its description and its listing in the repository are the same as normally done but as the user demands a service to be fulfilled, that particular request is represented as a Linear Time Logic (LTL) formula, the ‘Kripke’ in Figure 2.6, translates the WS description into Kripke styled
structure and then keeping temporal bound in check service composition can take place [HZ10].

[GvA18] maps WSC to Kripke structures as well, in an attempt to make sure that the branches that are running in parallel are not lost.

All the research work is done for WS composition considering temporal reasoning in some way or the other but the temporal aspect that is closely related to the domain of this thesis can be explained from an example in [HJXB07], the description of service repository might contain what and how the WSs provides a solution but without considering temporal(when) and spatial(where) it is provided the efficiency of the solution can not be guaranteed. For example, if someone plans to take a friend out for eating pizza and then to watch a movie in the cinema, on paper the service composition seems similar, first, find a pizza shop, order a pizza, get the package and make the payment and then look for a cinema and buy tickets but the temporal and spatial part of this service is very important, there is no point in finding a pizza shop that is far away from the cinema or buying tickets of a film that starts in few minutes and there is no time to buy or eat a pizza, so temporal aspect and spatial aspects of providing a composite service are as important as finding service for the user [HJXB07]. One of the important feature to look for in automated service composition is to have a composition that can change based on its current needs and responds to the changes in environment, one such example can be seen in [KLA16].
3 Background

As it is the case in all fields of research, Literature review about the topic and all the closely related aspects of it is massively important. Background and the vital information in relevance to personal assistants, services, service composition and some techniques that can be used for temporal reasoning is provided in this chapter.

3.1 Personal assistants

“An intelligent virtual assistant (IVA) or intelligent personal assistant (IPA) is a software agent that can perform tasks or services for an individual based on commands or questions. Sometimes the term ‘chatbot’ is used to refer to virtual assistants generally or specifically accessed by online chat. In some cases, online chat programs are exclusively for entertainment purposes. Some virtual assistants are able to interpret human speech and respond via synthesized voices”[Wik20c]. Like all areas where computers and automation software are introduced they make life easier on one hand and replace humans when tackling hectic problems on the other. In this scenario when Personal assistants (PA), Intelligent personal assistant (IPA), Software agents or even Virtual assistants whatever they might be called, are introduced and eventually replace the hectic job of a human personal assistant. People have employed others to take care of all the essential stuff that is going on around them from writing emails to reminding them of meetings to attend.

The usage of PAs is not confined to businessmen as it is more common than ever because of the readily available free versions provided by all the leading mobile phone and computer manufacturing companies. Just for the sake of some examples and mentioning the very few to ring some bells, Siri provided by Apple, Google Assistant by google, Cortana by Microsoft, Bixby by Samsung and many others that come free with the particular devices. Some very high-end PAs like Nina, Aido and Amazon echo are not free and provide very specific functionality. As mentioned above Amazon’s Alexa, Google’s assistant, Apple’s Siri and Microsoft’s Cortana are embedded in a device Amazon Echo, Google Home assistant, Apple iPhone and Windows PC respectively [Jon18].

Almost all PAs work in a similar fashion with natural language as an input in addition to using the graphical user interface (GUI) as an interactive input option provided by the software company. Even though a lot of work has been put in to recognize the natural language input for PAs but as every human being is different and thus providing a different challenge altogether resulting in usage of GUI. PAs can be divided into two categories firstly IPAs that perform typical concierge tasks based on the clicks on screen or voice input and secondly the smart PAs that can study, interpret and later improvise tasks that a user might perform based on its online history and location.
3.1.1 Daily life and family

Even though PAs are relatively new, they are frequently used in everyone’s daily life even if the particular person is not tech-savvy. Calling a particular contact person in the mobile phone, setting up an alarm, setting up meetings, making an appointment, adding a reminder, getting directions from the map are some of the features that are provided in the most basic versions of PAs present in all the top-end smartphones.

In Figure 3.1, as an example Siri is used, to wake it up ‘Hey Siri’ is used as voice input and it responds with the first screen on the left. The second screenshot shows the result of query ‘Will it rain tomorrow’ and third on the right shows the output when asked about having ‘any badminton booking’. As shown by the example, PAs on smart devices are just a voice call(‘Hey Siri’) away.

In the world where billions of devices can sense and collect data, later this data can be shared using a private or public Internet protocol, finally using this data as an understanding to make some actions would be the definition of the Internet of things (IoT) [PPSS16]. The progressive work done in the direction of IoT aided with PAs has transformed the dream of making a smart home into a reality. When home automation was introduced, initially it was not considered a success because it lacked the value for money as high setup cost could not be justified with minute improvement in the quality of life [CAS17]. Now, easy to set up devices are readily available in the market like Amazon echo paired with Alexa as the virtual PA. Sensors and smart devices capture the physical experience and transform it into meaningful information, it can be in the form of voice commands or gestures and can be used in controlling household appliances like fans, lights, air conditioners or even closing and opening of drapes [CAS17]. In recent years the increase in healthcare facilities around the world has led to the rise in life expectancy resulting in the need to take care of elderly, so smart PAs are widely used in providing a remote-controlled environment [CMP18].
3.1 Personal assistants

3.1.2 Context awareness of PAs

We refer to context as the interrelated conditions within the environment and all the relationships that exist between these conditions [BHS04]. Some of the conditions that might be of importance are time, location, environment, social setting, identity, history, tasks and device being used [BHS04]. The term context-awareness was used for mobile computers keeping in check the particular change in the environment of them, in general when a device is contextually aware it keeps track of its location and all the devices nearby [MAD00]. Another definition of context-awareness is “any information that can be used to characterize the situation of an entity, where an entity can be a person, place, or physical or computational object and further stating the use of context to provide task-relevant information and/or services to a user, wherever they may be” [AAD+99] [MAD00].

Even though groundbreaking technologies offer applications based on locality and context awareness, there still is a great degree of personalization needed to get the required task accomplished, Personalization means that every particular individual will be presented with information in a fashion that meets his or her needs but maybe in a different way to others [WHM+02]. In [WHM+02] personalization is divided into two parts, Filtering and Customization. Filtering uses the present time context of the user to decide which particular information is to be presented to the user. Customization is how to proceed with the presentation of that information, it can use several techniques employed by the end-user or the information provider, firstly the order in which assets(multimedia) are presented and secondly the selection of assets is done [WHM+02].

![Diagram](image)

**Figure 3.2:** PA with personalization [WHM+02]
3.1.3 Marketing PAs’

The advancement in IoT has made the human being’s relation with technology up, close and personal. As we have established up till now that PAs use voice commands to provide services to the user thus resulting in users avoiding a lot of advertisement, which in normal circumstances could not have been bypassed i.e typical on-screen advertisements [Jon18].

PAs are listening all the time, listening with a clear intent to serve the user to the best of their abilities. As PAs are learning constantly thus making the experience even more tailored to user needs and demands. The two main challenges marketers will face, firstly users will only get what they ask for instead of all the on-screen advertisements that were displayed previously, secondly the access to content will be done through PA in place of user-based online selection [Jon18].

As it can be seen in Figure 3.3 from [Bre19] a report published on smart speakers that have PAs, there is a huge potential to tap into marketing and running a successful business by adding intelligent PAs friendliness to your specific product or service. The market for AI is ever increasing, as it was at US 15.70 billion dollars in 2017 and predicted to grow up to US 300 billion dollars by 2026 [Sev19]. The most important aspect of AI is not to replace humans from the work environment but to aid them in a way making their life easier and productive, for example, AI can go through data and look at things humans might ignore and AI PAs can perform tasks to help in marketing and business. In [Sev19] four different ways are mentioned to make use of personal assistants to improve your business, Firstly PA apps can be used to automate workflows thus resulting in optimizing the business process, secondly using PAs or chatbots that can communicate with your current or
potential customers, thirdly all the services and products that your company is offering should be integrated with all the common PAs available in the market for example if a company provides travel facility then this facility should be bookable by a customer using apple Siri and lastly if it is not feasible to build on the currently available PAs for your product or service then the company should get a tailored PA of their own [Sev19].

3.1.4 PAs and Learning

In some research works, PAs are also referred to as agents, as their main aim is to achieve the completion of specific tasks to the best of ability but the way they go about it can be considerably different. In [KJES18], PA are divided into five different categories, simple reflex, goal-based, model-based reflex, utility-based and learning agents. Simple reflex agents use 'if then else’ rules to get the job done, on the other hand, model-based agents add a model to make better decisions, a pool of desirable results is stored in goal-based agents to pick the best fit result to match the goal, utility-based agents add further value to make decisions based on the level of desirability of particular goal [KJES18]. These four agents are well equipped to handle all the predictable environments but the problem arises when something happens out of the blue(unknown context or variable value) and then comes the learning agent which will adapt and change accordingly [KJES18].

In [MCF+99] a detailed experience about a learning PA is documented, here a PA called Calendar APprentice(CAP) is used by different people for at least sixteen months, the main theme behind the working of CAP was to use machine learning to assist it in making sensible decisions for the future. A lot of user interaction is required to make PAs future decisions and suggestions acceptable to a point. For example, a PA that has to sort emails according to the urgency, an email would be most urgent if it has the mention of a payment deadline but will lose its priority soon after the deadline is passed. Here in [MCF+99] a learning PA is introduced that is supposed to get better at predicting only if it has gone through enough learning experience, the approach taken can be summed up in three ways, firstly to provide a user-friendly interface, secondly to take every user interaction as a training example and lastly learn from all the training examples. Every night CAP is programmed to use its knowledge base acquired from training to make suggestive plans the next morning [MCF+99].

Figure 3.4: Usage tree for selecting meeting location [MCF+99]
Image of a decision tree in Figure 3.4 will be used to decide the location for the meeting depending on the time and department someone is from, the search process starts from the root and ends on the leaf node.

3.1.5 PAs for task and time management

Most of the PAs in the market perform general tasks and very few are tailored to deliver performance based on the narrow scope of expertise. One such PA (PExA(Project Execution Assistant)) that works on time management and task handling of workers in a company [MBB+07]. Time management as it appears in the name is to take care of the temporal aspects of the job like setting up a meeting or appointment time, task management, on the other hand, deals with how and when these tasks will be performed [MBB+07]. “PExA incorporates a significant body of sophisticated AI technologies for knowledge representation, reasoning (probabilistic and symbolic), planning, plan execution, agent coordination, adjustable autonomy, explanation, and learning. These technologies are integrated into a tightly coupled framework, drawing on a shared ontology and an agent architecture. This linkage has enabled a number of important capabilities within the system, including dynamic procedure learning, integrated task and calendar management, and real-time execution monitoring and prediction” [MBB+07].

3.1.6 Challenges and future of PAs

Like all technologies in the past, PAs are going through the development and evolutionary phase of their own, PAs evolved from saying ‘good morning’ and informing the user about the weather outside to perform mundane tasks, that when going through a hectic day would be ignored by humans. PAs have redefined the wheel when human-computer interaction is considered.

One of the major concerns of almost every sane mind while allowing PAs to gather as much personal information of the user as possible to give a personalized experience would result in privacy and security breaches. As for now, we can not anticipate how the companies, businesses or organizations collecting data from our PAs will be using it. As all the PA devices are always on and almost always listening and recording just to make it more intelligent but breaching privacy as well, so PAs should have detailed license agreement before its installation to make sure the user knows about all the data it lets PAs access at any given time, which is easier said than done.

A scenario in which a user is dropped to his home by a fully functioning autonomous car, after that the car departs for some servicing and vows to come by tomorrow morning to pick him for the airport, before going inside he picks up Amazon grocery packages delivered by drone, he then enters the house and after greetings, he is reminded of the planned travel tomorrow, seat and timing of flight bookings was not his headache because PA knows his preferences while air travel, upon unpacking the groceries he realised that PA has adjusted everything to travel size for aiding in luggage for travel and a lot more [Daw18]. Realistically speaking all the technologies and tasks mentioned above are being worked on and certainly, in the near future, we will have PA experience one very similar to this one.
3.2 Web Services

A web service can be defined as any piece of application, software or a document that is made available over the internet, it uses Extensible Markup Language (XML) as set standards of communication, web services strictly follow Transmission control protocol (TCP), Internet protocol (IP), Hypertext Transfer Protocol (HTTP), Hypertext Markup Language (HTML) and XML standards so that everyone with an internet connection can invoke and make use of them [Hug18]. Web services are not bound to be developed in any particular language (JAVA or Python) or operating system as long as they follow the set standards. For example, a python based web service that was developed using Flask when made available online can be accessed by someone using Java program on windows or java program on Ubuntu but with no difference at all.

Some single-line definitions for different components of web services, XML-RPC (Remote Procedure Call) is used to exchange data between client and server nodes, UDDI (Universal Description, Discovery and Integration) is a set of rules to be followed for publishing and discovery of web services, SOAP is a messaging protocol, API is an application programming interface and REST is a communication protocol used by APIs [Hug18].

<table>
<thead>
<tr>
<th>APIs</th>
<th>Web services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open source</td>
<td>Not open source</td>
</tr>
<tr>
<td>Light weight architecture</td>
<td>Uses SOAP so not a light weight architecture</td>
</tr>
<tr>
<td>Can use any form of communication</td>
<td>Only SOAP, XML-RPC and REST</td>
</tr>
<tr>
<td>Supports request/response headers</td>
<td>Only HTTP</td>
</tr>
<tr>
<td>Can be hosted with an app or IIS</td>
<td>Can only be hosted by IIS</td>
</tr>
</tbody>
</table>

Table 3.1: Highlighting APIs and Web services [Hug18]

![Figure 3.5: Interaction between client and service provider][1]

[1]: Figure 3.5: Interaction between client and service provider [Kre01]
Let’s consider an example to better understand the working of web services. It involves interaction between an accountant (client) and order processing system (web service). Firstly the client program collects all the information required for account registration into a SOAP message, later this message is sent to the web service using HTTP POST request, upon receiving, the web service assigns a new account number to this particular client and adds all the information into another SOAP message body to be sent to the client and finally, the client retrieves information about its new account from the response message [Poi20].

3.2.1 Discovery of semantic web services

Web services architecture explained in [Kre01] as shown in Figure 3.6 is composed of service registry, service requestor and service provider, the interactions between these are called publish, find and bind. The service provider has the software component of web service, in other terms, it has the main part, it makes a description of the service and publishes on the service registry for everyone to know that it is available to provide. Service requester, also known as the client, uses ‘find’ method to look for service in the registry that matches its needs and finally invokes the desired service by binding to the service provider.

Discovering desired services is the most vital part of the service-oriented system, with the consistently increasing number of service providers, providing astonishingly similar features also increase the difficulty for client-side to find the solution to its problem, this results in the need to have more futuristic routines to find web services [NKK10]. “Service advertisement, mediation, storage, request, matchmaking, negotiation and selection make up the service discovery system and while designing a semantic web service the key to its discovery is to follow the predefined set of criteria of compliance ”[NKK10].

OWL-S (Semantic Web Ontology Language), an approach that uses service profile and domain ontology to see if a match between published and requested service is possible, the approaches in OWL-S slightly vary from one another depending on QoS consideration and is employed matchmaking logical or non-logical, work in [PKPS02] does the matchmaking on IO description of services, [KFS06] and [KB08] introduce a new hybrid approach combining logic-based matching and token matrix [NKK10].
3.2 Web Services

WSMO (Web Service Modeling Ontology), an approach that considers required service from client-side, a goal to be fulfilled. WSMO is capable of handling heterogeneity that usually occurs in open environment [NKK10]. This feature is exploited by [KLL+05] and proposes a discovery mechanism to match client request. After experiments, it can be deduced that WSMO approach is very accurate but costly [NKK10]. WSDL-S (Web Service Description Language-Semantics) and SAWSDL (Semantic Annotation for WSDL) are a couple of other approaches that can be used for service discovery and matching, SAWSDL uses semantic annotations by enhancing the present WSDL framework [NKK10].

3.2.2 Web services discovery with QoS

Previously explained using Figure 3.6, the model used for service discovery. A new model proposed in [Ran03] deals with the quality of service provided, It also states that currently used UDDI model does not guarantee the quality of web page even after finding a suitable service by matching the service description of request and service repository. One of the major reasons to incorporate QoS parameter when looking for the desired service would be when two or more service description matches the request, the availability of QoS parameter of all the invokable services would make the client’s job easy [Ran03].

![Figure 3.7: New model for Web Service discovery [Ran03]](image)

The new model shown in Figure 3.7 works on the previously running web service discovery model and can fully function for clients that do not demand any quality of service. There are four roles instead of three, web service supplier and consumer as before, new UDDI registry and QoS certifier, as done before, the service is published on to the registry with its description but with the addition of QoS parameter, like before the consumer desires of a certain service and it gets a match from
3 Background

within the new UDDI registry but instead of, binding consumer to provider new UDDI registry it checks the QoS claims made by the provider using the QoS certifier and only after that binding takes place [Ran03].

3.2.3 Trust in web services

Trust in everyday human interactions can be a measure of how close one is to a particular human being. Just like humans, trustworthy web services should be invoked instead of services that do not provide the quality upon binding, which was promised in their description. Some researchers consider trust to be part of QoS but some consider it to be a totally different criterion.

Some roles are defined in [APC10] to measure or develop the trustworthiness of a web service. Firstly it states that trust is dynamic as it is always changing, trust depends on providers identity so that every interaction should be added to the identity, as trust can only be established upon having enough information thus it is demanded from the provider, a trusted third party can be involved to vouch for the provider and multiple trust establishing approaches can be used to further improve the quality [APC10].

3.2.4 RESTful versus Big web services

In [PZL08] a detailed comparison is made between RESTful WS and all the other commonly used WSs. The web application has a lot of architectural options for integration like sharing a database, having remote procedure calls (RPC), using a bus to communicate and batch file transfers but recently a new alternative ‘RESTful’ [Fie00] which gave a new dimension to RPC and is considered to be very simple to implement [PZL08].
3.3 Service composition

The article [PZL08] focuses on three different aspects while comparing, firstly how many decisions have to be made before invoking, secondly how many alternatives are available at the time and lastly the cost incurred due to development efforts [PZL08]. The conclusion derived after looking at the results suggests that if the lifespan is relatively long and requires high QoS then other Big WSs should be used but for tactical, easy to invoke and ad-hoc integration use RESTful WS [PZL08].

![Application Integration Styles](image)

**Figure 3.9: Application integration styles [PZL08]**

### 3.3 Service composition

The WS infrastructure explained in the previous section is sufficient to manage and provide interaction between clients and WS provider but sometimes a scenario might come up where one invocation leads to another or the client might need two different WSs to fulfil its requirement then Service Composition (SC) comes to play. For example, a person working at a company wants to send a package to another city and wants the packet to be insured as well, he selects ‘at-home pickup’ option from the online portal but this courier does not provide parcel insurance of their own, so using insurance from the third party and resulting in two services being combined to perform the required task. SC is sometimes called Composite services (CS) as well in [ACKM04]. The procedure of making a composite service either from elementary services or composite service is called SC [DS05].

<table>
<thead>
<tr>
<th>Models</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>What WS is and what not</td>
</tr>
<tr>
<td>Orchestration</td>
<td>Order in which WSs are invoked</td>
</tr>
<tr>
<td>Data access</td>
<td>Rules for data exchange between components</td>
</tr>
<tr>
<td>Service selection</td>
<td>Take care of binding</td>
</tr>
<tr>
<td>Transaction</td>
<td>Defines transactional semantics</td>
</tr>
<tr>
<td>Exception handling</td>
<td>To keep running and avoid failure</td>
</tr>
</tbody>
</table>

**Table 3.2: SC models [ACKM04]**

Online business logic, just like courier service above, might need a lot of WSs to be combined together, all these WSs or web applications can be in many different languages thus resulting in the need to have a standardized middle-ware, this leads to the development of Business Process...
3 Background

Execution Language for WS (BPEL4WS) [DVV03]. A middle-ware like this requires full detail of WS interfaces, how this WS will function and all the protocols it can support [DS05]. [ACKM04], describes six different models to be used for SC, described in the table 3.2. All the web services that allow business transactions, can provide users with much better experience if some sort of 'easy to use' instruction(language) can be made from the user side to the service provider [LAP06] [APY+02].

SC is a complex task in itself and the six main issues related to SC are coordination, transaction, context, conversation modelling, execution monitoring and the particular infrastructure of WS [DS05]. Using the WS model in Figure 3.6 it is easy to foresee interactions when one service is invoked but if composite services are to be invoked then some sort of coordination rules are needed to govern the sequence of interactions [CMC+03]. Coordination might layout rules of interaction but to give these interactions some surety then transaction protocol is needed, it works on the existence of coordination principle previously set [Pap03]. When talking about the context in WSs it is the personalized behaviour resulting from the collected information of the client, so rules and regulations are needed for context management [DS05]. SC can take place with centralized CS [SWZZ03] or distributed CS [SBDM02], in centralized, interaction takes place as the normal WS invocation because all the services are provided from a single server but in distributed, every CS has its own coordinator and they communicate to provide the client with the correct execution. Infrastructural changes like adding QoS to requirements(section 3.2,2) provided by service requester also lead to adding complexity to SC [DS05].

**Figure 3.10: WS composition infrastructure [PBM02]**

Different composition techniques have been discussed in [DS05], firstly static and dynamic SC, both related to the timing of SC, the static composition is done during the designing of the software system, all the components and how they compose a particular service is pre-decided and it works
3.3 Service composition

fine until the service it is composed of stays unchanged. If the particular business changes or turns to other services then the old ones are to be replaced and to secure the system from crashing runtime SC is needed, known as Dynamic SC [SWZZ03].

[OYP03b], proposes Model-driven SC, which is based on previously mentioned dynamic SC, it uses Unified Modelling Language (UML), UML helps to map one SC on the other standards. Rules for SC are modelled using Object Constraint Language and contains structural rules for scheduling services, data rules for data usage management, behavioural rules for event management, resource rule for resource management and finally exception rules to tackle unexpected events [DS05].

Figure 3.11: WS composition infrastructure based on [OYP03a]

As most of the times, the business process is created prior to rendering services according to the client’s needs but [Amb03] propose a different approach. it receives the client’s declarative requirements, the declarative SC consists of two phases, during first phase, the requirements are considered as a goal, that will be achieved and many plans are made to materialize the goal and in second phase one of the plans is chosen to work on. The architecture used to represent declarative SC is divided into three layers, conversation, functionality and data management layer [Amb03]. As the architecture described for declarative SC the present UDDI and service registry would not work so slight changes are to be made [DS05]. Other SC approaches that are discussed in [DS05] automated SC and context-based SC, where it is discussed that as different clients will have different machines (android phones, windows computer, McIntosh computers and iOS phones and tablets) to access the provided services thus changing the context for everyone should be managed using suitable frameworks [DS05].
3 Background

3.4 Techniques for temporal reasoning

Reasoning about time is one of the vital aspects of computer science, scheduling for tasks in embedded systems or real-time systems has been extensively worked on. There are many different ways time can be expressed and reasoned with like Allen’s Temporal intervals [All83], Linear Temporal Logic (LTL) and Computational Tree Logic (CTL) to name a few.

3.4.1 Allen’s Temporal intervals

Time can be taken as \( \pi\) (point interval) meaning that it starts and ends at the same time for example \( t=0.2s \) is a point in time. On the other hand, it can also be treated as an interval logic where time \( t \) starts at one point in time and continues until it aborts at another point in time. In [All83] introduced the famous Allen’s intervals which are widely used to represent time as intervals. The basic relations two intervals of time can have between them are depicted in Figure 3.12.

<table>
<thead>
<tr>
<th>Basic relation</th>
<th>Example</th>
<th>Endpoints</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x ) precedes ( y )</td>
<td>( \pi )</td>
<td>( x^+ &lt; y^- )</td>
</tr>
<tr>
<td>( y ) preceded by ( x )</td>
<td>( \pi^{-1} )</td>
<td>( y^- &lt; x^- &lt; x^+ ), ( x^+ &lt; y^+ )</td>
</tr>
<tr>
<td>( x ) meets ( y )</td>
<td>( \mu )</td>
<td>( x^+ = y^- )</td>
</tr>
<tr>
<td>( y ) met-by ( x )</td>
<td>( \mu^{-1} )</td>
<td>( x^- &gt; y^- ), ( x^+ &lt; y^+ )</td>
</tr>
<tr>
<td>( x ) overlaps ( y )</td>
<td>( \nu )</td>
<td>( x^- &gt; y^- ), ( x^+ &lt; y^+ )</td>
</tr>
<tr>
<td>( y ) overl.-by ( x )</td>
<td>( \nu^{-1} )</td>
<td>( x^- = y^- ), ( x^+ &lt; y^+ )</td>
</tr>
<tr>
<td>( x ) during ( y )</td>
<td>( \tau )</td>
<td>( x^- = y^- ), ( x^+ &lt; y^+ )</td>
</tr>
<tr>
<td>( y ) includes ( x )</td>
<td>( \tau^{-1} )</td>
<td>( x^- = y^- ), ( x^+ &lt; y^+ )</td>
</tr>
<tr>
<td>( x ) starts ( y )</td>
<td>( \gamma )</td>
<td>( x^- = y^- ), ( x^+ &lt; y^+ )</td>
</tr>
<tr>
<td>( y ) started by ( x )</td>
<td>( \gamma^{-1} )</td>
<td>( x^- = y^- ), ( x^+ &lt; y^+ )</td>
</tr>
<tr>
<td>( x ) finishes ( y )</td>
<td>( \delta )</td>
<td>( x^+ = y^- ), ( x^- &gt; y^- )</td>
</tr>
<tr>
<td>( y ) finished by ( x )</td>
<td>( \delta^{-1} )</td>
<td>( x^+ = y^- ), ( x^- &gt; y^- )</td>
</tr>
<tr>
<td>( x ) equals ( y )</td>
<td>( \equiv )</td>
<td>( x^- = y^- ), ( x^+ = y^+ )</td>
</tr>
</tbody>
</table>

Figure 3.12: Basic Allen interval relations [KJJ03]

Here \( x^- \) and \( x^+ \) represent an interval \( x \) which has its starting point as \( x^- \) and ending point at \( x^+ \), and \( x^- \) always come before \( x^+ \). The first two relations are ‘precedes’ and ‘preceded by’ if an interval of time is finished before the second interval starts then it can be said one precedes the other and vice versa from the perspective of the other interval (for example Tuesday precedes Thursday or Thursday is preceded by Tuesday). The third and fourth relationship is ‘meets’ and ‘met-by’, here the ending point of one interval is the same as the starting point of another interval, so it can be said that one interval meets the other and vice versa (for example Tuesday meet Wednesday or Wednesday is met
by Tuesday). The next two relations are 'overlaps' and 'overlapped by', here the starting point and ending point of one interval is before the respective starting point and ending point of the next and in addition to this the ending point of the first one has to be after the starting point of the next and then it can be said one overlaps the other (for example Alpino’s life overlaps the life of Brad Pitt). The next two relations are 'during' and 'includes', here one interval has the starting point after the starting point of the other but it ends before the second one and then it could be said the first interval is during the second(for example Tuesday is during a week). The next interval relationships are 'starts' and 'started by', here both the intervals have to have the same starting point and the ending point is not cared for, only then it can be said one starts the other and vice versa if looked from the other’s perspective (for example next Monday starts the new year). The next two relations are 'finishes' and 'finished by', here both the intervals have the same ending point without caring for the starting point, so it can be said one finishes the other and vice versa. The last and thirteenth relationship is when all the starting points and ending points of both the time intervals are exactly the same and then it is said the one equals the other.

One of the important things to note from the thirteen relationships is the first twelve are the inverses of each other and the last one is not. These relations can be described as distinct and exhaustive meaning that no pair of time intervals can be related to more than one Allen relations and at least one relation should describe them, all of these relations are asymmetric as just 'equals' is symmetric [All83].

Allen Intervals can be best explained with an example, “John was not in the room when I touched the switch to turn on the light, but John was in the room later when the light went out” [Dec03], the famous example used by Allen in [All83] as well. Let’s say ‘switch’ is the time when the switch was turned on, ‘room’ is the time when John was inside and ‘Light’ would be the total time light was turned on for. Now there can be multiple possible relations between these three intervals, shown in the table 3.3.

<table>
<thead>
<tr>
<th>Intervals</th>
<th>Relations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch and Light</td>
<td>can be 'overlapped' and 'meets' as well</td>
</tr>
<tr>
<td>Switch and Room</td>
<td>switch can be 'before', 'meets', 'met-by' or 'after' room</td>
</tr>
<tr>
<td>Light and Room</td>
<td>can be 'overlap', 'after' and 'during'</td>
</tr>
</tbody>
</table>

Table 3.3: Intervals and their possible relations

![Figure 3.13: Constraint graph and a possible solution [Dec03]](image-url)
This example can be expressed as a constraint graph ((a) in Figure 3.13 above), which has nodes as intervals or variables and edges represent constraints labelled with possible relations, the (b) in Figure 3.13 can be one of the possible solution graphs that correctly depicts this example [Dec03].

3.4.2 Linear Temporal Logic

When modal logic is considered, any proposition can be either true or false for example ’Is it Friday’, depending on the current day it can either be true or false [AV02]. “Linear Temporal Logic (LTL) is a modal logic representing time, LTL uses proportional variables (AP)” [Wik20b], some logic and modal operators briefly explained using the table 3.4.

<table>
<thead>
<tr>
<th>Operators</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Next</td>
</tr>
<tr>
<td>U</td>
<td>Until</td>
</tr>
<tr>
<td>W</td>
<td>Weak until</td>
</tr>
<tr>
<td>G</td>
<td>Globally or always</td>
</tr>
<tr>
<td>F</td>
<td>Future</td>
</tr>
<tr>
<td>⊨, □</td>
<td>Some future, all futures</td>
</tr>
<tr>
<td>⊤, ⊥</td>
<td>True, False</td>
</tr>
<tr>
<td>O</td>
<td>True in next moment of time</td>
</tr>
<tr>
<td>R</td>
<td>Release</td>
</tr>
<tr>
<td>M</td>
<td>Strong release</td>
</tr>
<tr>
<td>⊔, ∧, ¬</td>
<td>OR, AND, Negation</td>
</tr>
</tbody>
</table>

Table 3.4: Operators used in LTL [Art11] [Wik20b]

There can be two different ways to modal time, one can be the linear modal and the other is branched modal, Like all other logic, LTL formulae can be either satisfied or unsatisfied, a set of executions define the semantics [Art11].

Figure 3.14: LTL modal diagram example [Art11]

3.4.3 Computational Tree Logic

Computational tree logic (CTL) [CE82] is one of the branching tree logic as shown in Figure 3.15. When modelling in CTL, labelled state transition diagrams are used, consisting of nodes representing the current state of the system and edges representing possible transitions that can be taken from any particular state. When describing a CTL system state, as well as path properties, can
be used, state properties represent all True formulae at that particular state. As CTL is a branching tree logic this means if it represents time then the future is non-deterministic or can have many different paths and only one can be realised. At any given state a set of properties can be checked to make the next transition happen [Wik20a].

**Figure 3.15:** Branched modal diagram example [Art11]

State transitions and a resulting CTL modal can be best explained using a very simple example in Figure 3.16. As it can be seen on the left a three-state diagram with s0, s1 and s2 as possible states and all the edges representing transitions without showing any conditions, on the right a CTL modal that can be derived from the state diagram.

**Figure 3.16:** CTL example [Fou05]

The modelling of LTL and CTL can not be compared as they represent different properties at any given time [Fou05].
4 System design and execution flow

In this chapter the realisation of the idea behind this thesis is explained. It starts with the basic explanation of what were the options for us to choose from, for getting to a point where we could test few time-related queries to see how the service invocation might work. Then some insight on how to work with Application Programming Interface (API) in its simplest form for the reader and then we will see some explanation of the queries that were implemented along with the basic explanation of whats and how it is going on.

4.1 Service composition for a trip

The idea was to choose a scenario in which we can use some web services that were already available and get them working together in a single application. Logistic management service, hotel reservation and Travel management system were the three different options that were available for research.

As it can be seen from Figure 4.1 logistics management system comprises of a lot of other services that compose in a way to make sure that the service is delivered to the users. For the sake of an example, a user who intends to send a package from Stuttgart to Munich would have a lot of different options courier services in this regard, these courier services will make the system appear as a black box from the user point of view but the system actually comprises of other small systems or

![Figure 4.1: Logistic management system diagram](image-url)
procedures. Firstly the service provider and the user has to decide how the handover would take place, would the user drop it at a packet station or would demand a pickup from the user’s house. Then the second step would be to collect all the information related to the sender and receiver plus decision on whether the user wants the parcel to be insured or not and finally the payment is collected from the user.

One of the easiest ways to have temporal reasoning being implemented in an everyday scenario would have been possible in Logistics management system as we could have chosen a query that could have a packet being delivered in a way that aspect of time would have been involved, for example choosing to have an XYZ courier service that pickups from the sender’s home on weekends and deliver it to another courier service ABC for delivery to the actual address because XYZ does not provide services in the area of the receiver and ABC can not collect packets from customers on weekends. Unfortunately, as the logistic management services have very restrictive APIs it was very difficult to get some free working APIs that could be used to partially implement the temporal reasoning aspect of a query.

One of the other options was to have hotel management or booking system where it would be the same as before, would appear as a black box to the user but in reality, a lot of other services would combine to fulfil the user requests. Hotel booking system could have multiple sub-services like finding a hotel in the desired area, finding a room in some specific hotel, arranging food for the guests and finally the payment procedure.

Hotels’ booking system can also be used for a good example to implement a temporal reasoning example, let us say a person wanted to book a room in Stuttgart for five consecutive days and puts in the desired dates of stay but for some reason, all the hotels in the area might not have a single room available for straight five days and his query was rejected but if we have a system that could break the query into different options lets say first two days and then three days or like the first day...
4.1 Service composition for a trip

Figure 4.3: Temporal aspect involving hotel booking

in a single hotel and next four days in a different hotel. Even though it might not be very good for the experience of moving hotel to a hotel while on a single trip but implementing this temporal reasoning might help someone not miss an important trip.

Figure 4.4: Travel management system diagram

One other option was to look into a travel management system that includes all the different system like a flight booking system, train and bus booking system, hotel booking, taxi booking for users on a trip abroad, travel insurance and finally, like all systems involves the aspect of payment. As the main idea was to get a system reasonable enough to make it possible for some sort of temporal reasoning and then the automatic composition of services to reach the desired results.

One of the major branches of travel management system was flight booking and reservation, after intensively looking for free APIs that can be used to partially implement any of the above-mentioned systems, if not fully, the flight management and booking system was chosen to work around some specific queries that have something related to time and see how the system will work.
After choosing the flight management system, as it was the only system that had enough free to use available APIs that could make for this thesis to have some hands-on work done, then comes to the point to choose between the programming language that can be used for implementing all the framework for the working of specific APIs and all the other functionality needed can be implemented as well. Python was preferred over the use of JAVA as it had easy to use libraries that could work well with the system and make life easy for the developer, its not the JAVA does not have libraries that can be used to build this system but Python had very illustrative user manuals for specific libraries and even though after initially starting in JAVA, the switch seemed very appropriate. If only the execution time of the program was the deciding factor then probably C would have been a better choice than JAVA and Python both but given the ease of performing the same task in Python compared to C language led to the decision of making the application in Python, later it also helped in making the user interface for one of the queries.

For temporal reasoning, there can be two different ways it can be achieved, firstly we could have a query that will be entered by the user and after getting the query it can be given as an input to the temporal reasoner that will detect the temporal aspect of it and make the respective decision about how, when and which service to be invoked or if there is a need for some sort of service composition then how this composition will take place to cater the important constraint of time. The second method that can be used is to let the user enter the query and after the query is received by the application, it proceeds in the normal way as usual but after receiving the results it should not display the results to the user directly but let them go thought the temporal reasoner and see how the results can be improved if possible. Both the techniques for having a temporal reasoner before the services are invoked and after can be of importance.

![Possible temporal reasoner flow diagram](image)

As shown in Figure 4.5, both the options can be chosen for reasoning for this thesis, in the first option the drawback is that a lot of different meanings can be understood from the user query but as the decision is made, then the final result can be displayed to the user. In the second option when the service invocation results are received, that is used as an input to the reasoner, here the problem to look for is the amount of data that is to be processed, as it can be a massive chunk of JSON (JavaScript Object Notation) file to use and then make decisions that can be temporally effective.
4.2 Getting started with basic APIs

To get familiarity with how the APIs work normally and what are the general procedures that are used to get them working. The basic idea is simple, firstly any service or API provider has some sort of documentation, that is provided online has to be downloaded and read for reference or at least important parts of how can a third-party developer integrate the provided service in its application. All the APIs are not free to use and this was one of the most hectic parts of this thesis, as finding an API that allows unlimited or even partial access were very rare. Some APIs like IATA codes finder that finds a number of airports and their respective airport codes if given with the city name is free to use as the provider deems this service to be of little to no value when considering revenue but on the other hand, if one takes the example of DHL (courier service), they do not provide any information related to their API, as DHL’s API is one of the biggest assets for them. Some APIs like Expedia (provide hotel and travel reservations) do allow for users to access or use their respective APIs in third party applications after joining them as affiliate partners and paying them a fixed fee for the complete period of subscription.

Listing 4.1 The code shows authentication requirements.

```python
account_sid = 'AC312f1bf8e09017293cab01c6b20de3b'
authentication_token = '48355a532467e0940c3485ef8b0c244e'
client = Client(account_sid, authentication_token)
```

For example in the code shown in listing 4.1, It is for one of the APIs that gave four hundred free accesses per month and after that, it will be charged, so to keep track of authentic login, a unique account identification number is allotted to the registered user and authentication token is provided as well. In the end, both these identifiers are added to the header part of the communication message that will be used to interact with the host application and only upon approval further access will be provided. Most of the APIs used in this thesis were of this type that allowed few hundred accesses per month, to cater this two different steps can be taken, firstly a counter can be used to keep track of the number of API messages or different accounts can be made with the host so that once the quota is reached another ID can be used to continue the access.

Listing 4.2 The code shows the libraries that are to be used

```python
import requests
import response
import json
import pprint
```

The code shown in the listing 4.2 is the initial chunk of lines that are very important as these are used to import or install additional libraries that will be used to get the APIs working. The first library of ‘requests’ and the second of ‘response’ are the vital message libraries that are HTTP methods for communication between two entities. ‘Request’ is used to send a message from one application to another and the reply that is received by the sender is called the response. The next
library is JSON, which is used for the understanding of the response that is received, as almost all
the responses when some type of information is asked from the host is in JSON format and lastly,
`pprint` is for printing.

**Listing 4.3** The code shows the basic implementation of an API in Python

```python
request_made = requests.get('http://api.open-notify.org')

def jprint(obj):
    # for the formatting of JSON files
    text = json.dumps(obj, sort_keys=True, indent=6)
    print(text)

response = requests.get('http://api.open-notify.org/astros.json')
jprint(response.json())

response_json = response.json()

# for the purpose of printing people who are in space
print("Number of people in space: ", response_json["number"])  
# for the purpose of printing their names
for x in people_json["people"]:  
    print(x["name"])
```

In listing 4.3, the most basic use case of implementing an API is shown for the understanding of
how it is done, this code is in a Python file written in a local environment but it is accessing data
that is provided by an API. One of the first steps to do is to get the documentation provided by the
API host and go through the part where it mentions 'how to make a header that is recognizable and
will get the access to the system', after getting the access using the request command the propose of
actually accessing it can be performed, like in this small example as shown in listing 4.4, some sort
of data is requested using ‘get’ command and it is stored in 'people', as the data retrieved from the
web application is mostly in JSON format then that data is to be parsed in a meaningful way. This
short example uses an online application that provides the names and number of people that are
currently in space at this moment.

**Listing 4.4** The code shows the request.get command in usage

```python
request = requests.get('http://api.open-notify.org')
people = requests.get('http://api.open-notify.org/astros.json')
```

The listing 4.6 shows the JSON response from the output of this application after running it in
Python console. Then listing 4.4 can be used to retrieve information from the received JSON file,
this can be very useful when doing temporal reasoning after getting the JSON response, here the
names of the astronauts in space are given along with their craft but the listing 4.5 can be used to
extract just the relevant information, for example, their names as shown in listing 4.7.
### Listing 4.5 How to retrieve relevant information from the received JSON

```python
for x in people_json['people']:
    print(x['name'])
```

### Listing 4.6 The code shows the JSON response

```json
{
    "message": "success",
    "number": 5,
    "people": [
        {
            "craft": "ISS",
            "name": "Chris Cassidy"
        },
        {
            "craft": "ISS",
            "name": "Anatoly Ivanishin"
        },
        {
            "craft": "ISS",
            "name": "Ivan Vagner"
        },
        {
            "craft": "ISS",
            "name": "Doug Hurley"
        },
        {
            "craft": "ISS",
            "name": "Bob Behnken"
        }
    ]
}
```

### Listing 4.7 The required information after using listing 4.6

- Number of people in space: 5
- Chris Cassidy
- Anatoly Ivanishin
- Ivan Vagner
- Doug Hurley
- Bob Behnken
4.3 Case study’s block diagram

To briefly explain the flow for this thesis, a block diagram is shown, making it easy to understand. As this thesis aims to work in the direction of personal assistants taking user input in the form of voice and then performing tasks based upon the understanding of the user requirements, but here instead of Natural Language Processing (NLP) queries are taken in the form of user input to the prototype application, for one of the queries a user interface is also designed to have a clearer picture of how the input is given to the system and how it will be displayed and this will be explained later in the chapter. As the input is given to the wrapper, the first step is to have some sort of temporal reasoning on it, this will be explained in the next subsection of 'Sample queries', later finding places and IATA codes (unique airport identifiers) are given to the Skyscanner and FlightData APIs for searching for the desired flight, if there is one available while developing this application, all the flight searching APIs used to return a lot of data for cities like London to New York as there were many flights and this was one of the deciding factors to make temporal reasoning before we invoke any service but now during the pandemic, few flights are flying so the data is limited and can be easy to visualize as well even in JSON format. For better understanding the data has to be in a presentable format and to do that a converter is used that converts JSON files in tables, these tables are the output for the user and in addition to this a Short Message Service (SMS) can be sent to the user for notification or confirmation, this SMS is not a vital part of the thesis.

![Block Diagram](image)

**Figure 4.6:** Complete block diagram for the thesis
4.4 Sample queries

In this section, all the queries that were implemented successfully are explained, it also contains few of the important parts of the application to put some light on how things were done and all the results and outputs from these queries will be displayed in the next chapter by the name of evaluation.

4.4.1 Query: One way flight

The first query to be implemented in this thesis was a simple one, which states for example ’The user wants to book a flight from one place to another on a specific date and this flight is one way’, here the date, and both the destination and origin of the flight can vary according to the user.

One of the first things to tell about the implementation of these queries would be the different libraries that were used throughout this thesis. As it can be seen from listing 4.8, requests and JSON are used similar to the way mentioned in the previous subsection, ‘datetime’ is used to understand the date the user wants to book a flight on, ‘datetime’ library is useful in making sure about the exact date for the request sent to the host API, pandas is used to make the received JSON data into something that is visually understandable and lastly, the most important library is ‘urllib’, this helps in parsing or editing the host URL so that the required set of information can be provided to the host and in return, the valid information is received. Almost all the queries use the same set of libraries.

Listing 4.8 The required libraries in Python

```python
import requests
import json
import datetime
import pandas as pd
import urllib
from urllib.parse import urlparse
```

The next step to do was to get the received response into something meaningful, as it can be seen in the listing 4.9 that once the date of an outbound flight, city of origin and destination is entered and all the data is received as a single line, this is not even one-fifth the data that has been received as most of the data is replaced by the dots at the end but the point was to show how it will be converted into understandable form making a custom printing function that will serve this purpose.

Listing 4.9 The function made for custom printing

```python
def jprint(obj):
    # for formatting JSON files to understandable form
    text = json.dumps(obj, sort_keys=True, indent=4)
    print(text)
```
4 System design and execution flow

Listing 4.10 The response with out custom printing

Please enter the date you want to fly out on in YYYY-MM-DD format 2020-12-10

Where do you want to fly from LHR-sky

Where do you want to fly to JFK-sky

{ "Routes": [], "Quotes": [{ "QuoteId": 1, "MinPrice": 347.0, "Direct": true, "OutboundLeg": { "CarrierIds": [857], "OriginId": 60987, "DestinationId": 60987, "DepartureDate": "2020-12-01T00:00:00" }, "QuoteDateTime": "2020-06-22T08:30:00" }, { "QuoteId": 2, "MinPrice": 258.0, "Direct": false, "OutboundLeg": { "CarrierIds": [1760], "OriginId": 60987, "DestinationId": 60987, "DepartureDate": "2020-12-01T00:00:00" }, "QuoteDateTime": "2020-06-22T08:03:00" }, { "QuoteId": 3, "MinPrice": 1667.0, "Direct": true, "OutboundLeg": { "CarrierIds": [838], "OriginId": 60987, "DestinationId": 60987, "DepartureDate": "2020-12-02T00:00:00" }, "QuoteDateTime": "2020-06-22T06:29:00" }, { "QuoteId": 4, "MinPrice": 219.0, "Direct": false, "OutboundLeg": { "CarrierIds": [881], "OriginId": 60987, "DestinationId": 60987, "DepartureDate": "2020-12-02T00:00:00" }, "QuoteDateTime": "2020-06-22T06:29:00" }........]}

Listing 4.11 Relatively understandable form

Please enter the date you want to fly out on in YYYY-MM-DD format 2020-12-10

Where do you want to fly from LHR-sky

Where do you want to fly to JFK-sky

{ "Quotes": [ "Direct": true, "MinPrice": 374.0, "OutboundLeg": { "CarrierIds": [881], "DepartureDate": "2020-12-10T00:00:00", "DestinationId": 60987, "OriginId": 60987 }, "QuoteDateTime": "2020-06-21T21:07:00", "QuoteId": 19 } ]}
4.4 Sample queries

4.4.2 Query : Return flight

The second query was similar to the first one but has an added complexity, it states for example 'The user wants to book a flight from one place to another on a specific date and want a return flight to the city of origin on a specific date', here the date of departure, arrival and both the destination and origin can vary according to the user.

In this query the libraries used are the same as before, the only difference is additional information about the date of the return flight. The input method as seen from the listing 4.12 is almost the same as before with the collection of outbound date as well as the inbound date. The date is converted in the string format to make sure that it can be added to the URL which was accessed to retrieve information from the host website.

Listing 4.12 The code that allows input of date in specific format

```python
date_entry = input('Please enter the date you want to fly out on in YYYY-MM-DD format ')  
year, month, day = map(int, date_entry.split('-'))  
date_outbound = datetime.date(year, month, day)  
dateTimeObj = date_outbound  
date_outbound_Str = dateTimeObj.strftime("%Y-%m-%d")

date_entry = input('Please enter the date you want to return back on in YYYY-MM-DD format')  
year, month, day = map(int, date_entry.split('-'))  
date_inbound = datetime.date(year, month, day)  
dateTimeObj = date_inbound  
date_inbound_Str = dateTimeObj.strftime("%Y-%m-%d")
```

4.4.3 Query : To find cheapest routes on specific dates

The third query is a simple one as it uses the first query as an example to work on, but it works in a slightly different way as it finds the cheapest route from one place to another, it works on finding the cheapest ticket from all the searched tickets thus making sure that information can be retrieved from the resulting JSON files.

Listing 4.13 The code that allows URL encoding

```python

url = mainurl + urllib.parse.quote_plus(fly_from) +'/'+ urllib.parse.quote_plus(fly_to) +'/'+ urllib.parse.quote_plus(date_outbound_Str)

querystring = {"inboundpartialdate":date_inbound}
```
One of the things that are common in all the queries is how the URL encoding is done in a way to meet the specific requirements, as with every query some part of the URL or some part of the header that is used to access the host API is changed. The headers can contain important identification information that will be the very first interaction between this application and host and only after authorization can the rest of the communication be made possible. The listing 4.14 shows one of the URL encoding that was done for a query and all the other queries have something similar to this. 'fly-to' and 'fly-from' are variables that can collect input for the origin and destination cities, 'mainurl' is the part of the host URL which will always stay the same no matter what, 'url' is a string-based variable that is used to make the final URL that will be used to invoke the service, 'fly-from' and 'fly-to', 'outbound date' is parsed using the 'urllib', the inbound date is added as one of the strings that can be optionally added to the 'url'.

**Listing 4.14** The inputs taken for origin and destination cities

```python
fly_from = input('Where do you want to fly from ')  
fly_to = input('Where do you want to fly to ')  
```

4.4.4 Query : For enlisting all flights in a month

The fourth query is a little different as it states 'Enlist all possible flights from one place to another within a specific month', this query has a different aspect of time involved than the previous ones as it does not have a particular date to give to the APIs but an entire month. The collection of data from the user is done in the same way as done in the previous queries, the user has the option to select any city of origin and destination as well as the month to find the possible flights, the result of this as well as all the other queries will be shown in next chapter. This query takes in the month as an input and then displays all the possible flights in that month. The code segment shown in listing 4.15 was used to automatically compose the desired results.

**Listing 4.15** This code is used for selection of right month based on user request

```python
today = date.today()  
if date_entry in ['January', 'Jan', 'Jan', 'january']:  
mymonth = 1  
elif date_entry in ['February', 'feb', 'Feb', 'february']:  
mymonth = 2  
elif date_entry in ['March', 'mar', 'Mar', 'march']:  
mymonth = 3  
# same for all the other months but code not shown  
if mymonth > today.month :  
    myyear = today.year  
else: myyear = today.year + 1  
date_outbound = datetime.date(myyear, mymonth, myday)  
dateTimeObj = date_outbound  
date_outbound_Str = dateTimeObj.strftime("%Y-%m-%d")
```
Variable 'today' that can be seen in the listing 4.15 is used to get the current date, month and year, as the user enters the required month to find the flight in, the application checks if the asked month is after the current month or not and if it is after that then the month is selected as per the request and the year for query stays the same. If the month asked is before the current month, for example, if a user asks now in June to find me all possible flights in January then the year should be the next year(2021) and not '2020'.

One of the important things to do was to get the response into a format as understandable as possible, the Figure 4.7 shows on the right, the small part of the information displayed in a table format that was converted from JSON file on the left. This is just a very small example just to show how things can look different and information can make more sense when put in a presentable format as in reality when working with this thesis the usually resulted file consists of several hundred lines and that can make it very difficult to understand, so converting JSON to the table makes it very efficient.

```json
{
    "message": "success",
    "number": 5,
    "people": [
        {
            "craft": "ISS",
            "name": "Chris Cassidy"
        },
        {
            "craft": "ISS",
            "name": "Anatoly Ivanishin"
        },
        {
            "craft": "ISS",
            "name": "Ivan Vagner"
        },
        {
            "craft": "ISS",
            "name": "Doug Hurley"
        },
        {
            "craft": "ISS",
            "name": "Bob Behnken"
        }
    ]
}
```

**Figure 4.7:** Display of sample JSON file on left and the converted table from this on right

### 4.4.5 Query: Enlisting all possible flights in next week

The fifth query is somewhat similar to the fourth one as it also requests to enlist a chunk of flights but the time period is within a week. This query would be stated as something like ‘find all possible flights from one place to another in next week’. In this query, the week in which the user wants to travel will be changeable as well as the cities like in every other query. The libraries used in this query are the same as before, the application would ask about the city of origin and destination to look for all possible flights in the next week.
4 System design and execution flow

Listing 4.16 This code is used for selection of right week based on user request

```python
import calendar
import datetime
import calendar

today = date.today()
print("Today's date:", today)

yy = today.year
mm = today.month

print(calendar.month(yy, mm))

from datetime import datetime, timedelta

days_to_new_week = 8 - today.isoweekday()
new_date_start = today.day + days_to_new_week

The next week depends on the day the user makes the actual request, every week starts from Monday and ends on Sunday, as the input previously worked with dates in a specific format but now it is just the query and it automatically has to consider the current date and calculate the date for the coming Monday, for this 'today.isoweekday()' is used which tells the current number of the day in a week, for example, Wednesday would result in 3, so to calculate the new date we needed the exact number of days to the start of next week so 'day-to-next-week' is calculated. Then in a loop of seven(representing seven days of the next week) the query to find a flight on each specific day is done to get the desired results.

Listing 4.17 This code is used for selection of right week based on user request

```python
for weekday in range(0, 7):
    year = 2020
    month = int(today.month)
    day = int(new_date_start + weekday)

date_outbound = datetime.date(year, month, day)

print(date_outbound)

url = mainurl + urllib.parse.quote_plus(fly_from) + '/' + urllib.parse.quote_plus(fly_to) + '/' + urllib.parse.quote_plus(date_outbound_str)
querystring = {'inboundpartialdate': date_inbound}
headers = {
    'x-rapidapi-host': "skyscanner-skyscanner-flight-search-v1.p.rapidapi.com",
    'x-rapidapi-key': "b3f0a2056emshc4e2c2becb84760p1f0a00jsn7bb751d3723c"
}
response = requests.request("GET", url, headers=headers, params=querystring)
jprint(response.json())
```
4.4.6 Query: Possible flights on a specific day next week

The sixth query is the most complex one but it works on the previous few queries so it can be accomplished, this query can be stated for example 'find a flight from one place to another on next Wednesday'. Here a lot of things can vary according to the will of the user, for example, the first thing that would be different is the relative 'next week' as it will depend on the day and week of the request being made and of course the city of origin and destination can vary accordingly.

The user enters the query as 'find me a flight on Wednesday next week', as the queries take integer dates as an input to work but here the input is a day(Wednesday), so application computes the current day(on which the query was made) and respectively calculates the desired date for the query to function properly. Here in this query, any day can be asked from the next week according to the will of the user. Days to next week are calculated as done in the previous query, here the query differs from the previous one it finds the flights for a specific day and date next week.

**Listing 4.18** This code is used for selection of right week and day based on user request

```python
    days_to_new_week = 8 - today.isoweekday()
    new_date_start = today.day.__add__(days_to_new_week)

    date_entry = input('Please enter the day you want to find the flight for in next week ‘)

    if date_entry in ['monday', 'mon', 'Monday', 'Mon']:
        day_number = 0
    elif date_entry in ['tuesday', 'tue', 'Tuesday', 'Tue']:
        day_number = 1
    elif date_entry in ['wednesday', 'wed', 'Wednesday', 'Wed']:
        day_number = 2
    elif date_entry in ['thursday', 'thu', 'Thursday', 'Thu']:
        day_number = 3
    elif date_entry in ['friday', 'fri', 'Friday', 'Fri']:
        day_number = 4
    elif date_entry in ['saturday', 'sat', 'Saturday', 'Sat']:
        day_number = 5
    elif date_entry in ['sunday', 'sun', 'Sunday', 'Sun']:
        day_number = 6

    new_date_search = new_date_start + day_number
```

4.5 User Interface for a query

As shown in Figure 4.8, the user interface for one of the queries, as this was not the integral part of the thesis but to have a view of how this can be implemented in a web application only one of the query was used. This is the front end of the second query where the user wants to book a flight from one city to another on a specific date and want a return flight to the city of origin on another date.
Whenever the users run the system, they will experience a display of a window shown in Figure 4.8. This window is called User Interface generally. The user interface for this system is designed in a way that it needs some inputs which are the information known by the user. In this interface, the input fields are From, To, Outbound and Inbound. From defines the departure airport from which the user wants to fly from, To defines the arrival airport in which the user is intending to go, Outbound defines the date on which the user wants to travel from one place to the other and Inbound is the date on which the user is planning to return. The user must fill From, To and Outbound fields while the Inbound field is optional since it is not necessary that everyone will look for a return flight. An example of filled fields is shown in Figure 2. Hence, the necessary information is taken from the user and if all the requirements of particular fields are met and after this, if the user presses the Submit button, all the information will be sent to the back-end for further processing with the help of JQuery.
The user interface is implemented by using the Flask framework in Python environment which makes it easy to comprehend and implement the interface functions. This particular user interface has been deployed on localhost which makes things easier for testing environment. The input data taken from the user input on localhost is processed in python with the help of the Flask framework after the user presses the Submit button on user interface followed by the filled fields in the interface.

![Figure 4.9: GUI with example data](image)

The user interface page is an HTML page which is displayed on a web page using an HTML template. The Flask framework is deploying a web app on the local host and the web page is displayed with the HTML template which is user friendly.

![Figure 4.10: Possible temporal reasoner flow diagram](image)
5 Implementation of use cases

In this chapter, one simple example of every query will be given and the results that will be compiled will be given as well, as in some cases the results are long, only few will be shown to keep things simple and understandable. All the results are based on the system design taken from the previous chapter.

5.1 Query: One way flight

One of the examples for this query in words can be 'Please find me a flight on 2020-12-01 from London to New York.' The input for this one would look like in Figure 5.1, with having entered the desired date to fly from London Heathrow to John F. Kennedy in New York.

```
Please enter the date you want to fly out on in YYYY-MM-DD format: 2020-12-01
Where do you want to fly from: LHR-sky
Where do you want to fly to: JFK-sky
```

**Figure 5.1:** Example input for query 1

```
"Places": [
  {
    "CityId": "NYCA",
    "CityName": "New York",
    "CountryName": "United States",
    "IataCode": "JFK",
    "Name": "New York John F. Kennedy",
    "PlaceId": 60987,
    "SkyscannerCode": "JFK",
    "Type": "Station"
  },
  {
    "CityId": "LOND",
    "CityName": "London",
    "CountryName": "United Kingdom",
    "IataCode": "LHR",
    "Name": "London Heathrow",
    "PlaceId": 65688,
    "SkyscannerCode": "LHR",
    "Type": "Station"
  }
],
"Quotes": [
  {
    "Direct": false,
    "MinPrice": 225.0,
    "OutboundLeg": {
      "CarrierIds": [
        "881"
      ],
      "DepartureDate": "2020-12-01T00:00:00",
      "DestinationId": 60987,
      "OriginId": 65698,
      "QuoteDateTime": "2020-06-22T21:42:00",
      "QuoteId": 1
    }
  },
  {
    "Direct": true,
    "MinPrice": 1668.0,
    "OutboundLeg": {
      "CarrierIds": [
        "888"
      ],
      "DepartureDate": "2020-12-01T00:00:00",
      "DestinationId": 60987,
      "OriginId": 65698,
      "QuoteDateTime": "2020-06-22T16:53:00",
      "QuoteId": 2
    }
  }
]
```

**Figure 5.2:** Example output in json format
The results when displayed in JSON format are not easy to understand, so they were converted to table format for better understanding, all the following queries will have results shown in table format only. The output displays all the different carriers that are flying from these two cities along with giving all the other details as well like 'Currencies' displays the currency that was selected to display the ticket prices in, it can be changed according to the user, 'Places' show all the details acquired about both the cities of origin as well as a destination city and finally 'Quotes' show the actual options that a user can select.

5.2 Query: Return flight

One of the examples for this query can be 'Please find me a flight on 2020-12-01 from Berlin to London and back on 2020-12-05'. In this query the user asks for the return flight and the results will be similar to the previous query but just with the return flight also searched and displayed as it can be seen from the Figures 5.5 and 5.6. The first output figure shows the flight from Berlin to London and the next shows the flight on the desired date in the opposite direction.
One of the examples for this specific query can be 'Please find me the two cheapest flights from Dubai to London on 2020-12-05'. The Figure 5.7 the possible input that will be taken by the application and then Figure 5.8 shows the two cheapest tickets from the desired origin to destination, this information can be seen under the heading of 'MinPrice' column in the table. The first section of the table gives information about both the cities like in this case Dubai and London.
5 Implementation of use cases

Figure 5.7: The example input for this query

![Example Input](image)

Figure 5.8: The example output for this query

![Example Output](image)

5.4 Query : For enlisting all flights in a month

One of the possible examples for this type of query can be 'Please find me all possible flights from London to New York throughout the month of December', in this query the user can ask for any month depending on personal requirements and similar to all other queries the city of origin and destination can vary as well. The Figure 5.9 shows the sample input for one such example and later some of the initial results for this query are shown as there are like more than seventy flights that are a result of this query and can not be shown here but the initial few can be shown.

Figure 5.9: The example input for this query
5.4 Query: For enlisting all flights in a month

As it can be seen from Figure 5.10 there are two flights on first of December and then there are other flights on later dates, these dates can be seen from one of the fields named ‘Departure Date’ in ‘Outbound Leg’ column.

**Figure 5.10:** The example output for this query
5 Implementation of use cases

5.5 Query : Enlisting all possible flights in next week

One of the possible query statements here could be ‘Please find all possible flights from Berlin to Istanbul in next week’, for this the sample input that can be taken in this application can be seen in Figure 5.11 as it asks for only the city of origin and destination. All the other information that can be seen like the current date and the calendar is there for the reference only.

Figure 5.11: The example input for this query

| Places | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| CityId | CityName | CountryName | IataCode | Name | PlaceId | SkyscannerCode | Type |
| ISTA | Istanbul | Turkey | IST | Istanbul | 59316 | IST | Station |
| BERL | Berlin | Germany | TXL | Berlin Tegel | 84692 | TXL | Station |

<p>| Quotes | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|</p>
<table>
<thead>
<tr>
<th>Direct</th>
<th>MinPrice</th>
<th>OutboundLeg</th>
<th>QuoteDateTime</th>
<th>QuoteId</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>174</td>
<td>1953</td>
<td>2020-06-22T18:35:00</td>
<td>1</td>
</tr>
<tr>
<td>true</td>
<td>108</td>
<td>1755</td>
<td>2020-06-22T18:35:00</td>
<td>2</td>
</tr>
<tr>
<td>false</td>
<td>172</td>
<td>1939</td>
<td>2020-06-23T16:34:00</td>
<td>1</td>
</tr>
<tr>
<td>true</td>
<td>225</td>
<td>1755</td>
<td>2020-06-23T16:34:00</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 5.12: The example output for this query
5.6 Query : Possible flights on a specific day next week

For this particular query, one of the example statements can be 'Please find me a flight on Tuesday next week', the query in the example was made on 2020-06-23, so after calculation, the application would know the next week as well as the date on Tuesday during the next week which turns out to be 2020-06-30 and as it can be seen from the sample output from this particular query show flights on 2020-06-30 in Figure 5.14.

Figure 5.13: The example input for this query

Figure 5.14: The example output for this query
6 Conclusion and future work

6.1 Conclusion

One of the most promising aspects of the Internet is that it can make the usual day to day businesses online, in this age and especially after the Pandemic of 2020 E-businesses will be on the rise, in fact, Amazon which is considered to be the global leaders when it comes to online shopping and selling has made one of the highest profits in this time, stating that they have made 1.25 times the sales during the same time in last year [RS20] . As E-commerce is on the rise, it, in turn, gives rise to the usage of technology in a different way to the past, thus leading to the use of personal assistants and resulting in making users make verbal queries to their smart devices.

This thesis works in a direction that would help to make some good temporal based choices when selecting to compose web services, the first phase of the thesis was to familiarize with the state of the art research that was done in this particular field and then it was to get to know the importance of web services, how these web services were being used with the personal assistants and finally looking for different options that can be used to show some sort of calculations based on time. First few chapters like 'State of the art' and 'Background'’ were mostly the learning part and then 'Implementation' has all that was done to make some queries that had the aspect of time in them. Later in 'Evaluation’ part, these queries were shown with some sort of sample data that the application might come up with after understanding the natural language input from the user.

In this thesis several queries were developed that would run in a Python-based wrapper, all the queries had some sort of time-based decisions to be made and based on those decisions the specific web services were invoked and after the invocation, all the results were compiled and displayed in an understandable format and in addition to this, a user interface was also developed for one of the queries to get to know the working of a web-based application.

6.2 Future work

Nowadays, all the projects, either hands-on or just research-based projects, after their scheduled conclusion time still have some room for improvement thus giving an open window for the future work to take place. For this thesis future work can be done with considering the following aspects.

- Natural language processing

As for this thesis, the main idea was that the user of this application would be giving all the instructions as verbal commands (Natural language input) and then that input would be converted to some understandable form for the application but here to keep things simple the input was made
manually thus minimizing the scope of this thesis. In future, a system that can take NLP input and make service composition and reasoning based on those would be one step closer to getting an application ready to be used in the market.

- **Getting paid services**

One of the things that held back the working of this thesis was to find free APIs or at least those APIs that provide a full version to be used for researchers, as all the APIs that have a lot of data in their respective databases do not share the complete data with free users thus limiting the data that can be used to see how good the results are compared to other services, for example taking Skyscanner into consideration when one query is entered into this system, that resulted in possible flights that are only a fraction of all those possible on the official website of Skyscanner.

- **Using other forms of reasoning**

The other thing that can be added to this thesis and can further enhance the research is to use other types of reasoning as well, within temporal reasoning other types of Linear temporal reasoning as well and computational tree logic can be tested as well. To broaden the research aspect, the shift from temporal to spacial reasoning [MB07] can be made, service compositions with the added aspect of space and time will be one step closer to the aim of getting complex web service composition made easy.
Bibliography


Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature