Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelor Thesis

Service composition in the domain of travelling

Venilin Pirev

Course of Study:	Medieninformatik
Examiner:	Prof. Dr. Marco Aiello
Supervisor:	Prof. Dr. Marco Aiello
Commenced:	March 30, 2021
Completed:	September 21, 2021

Abstract

Web services and personal assistants are used more and more in our everyday life. Unfortunately we are very limited, when it comes to using them in a domain of travelling. Web service compositions solve some of the problems the user faces, but also creates others. Most of the existing web service compositions just aggregate different services, which makes it hard to deal with temporal or spatial-based variables. This affects not only the run-time of the service, but also the delivered results to the users. The solution we propose uses different appropriate temporal models with every service, in order to avoid possible errors and undesired results. It also uses different spatial models in this regard to create a seamless communication between the service and the different Application Programming Interfaces (APIs). Furthermore the proposed solution will compose the different responses in order to deliver a result, according to the users specification and requirements. The evaluation in this thesis provides evidence of the significantly lower run-time of the solution, which in most cases is twice as low compared to using conventional methods.

Contents

List of Figures

List of Algorithms

Acronyms

API Application Programming Interface. 13

APIs Application Programming Interfaces. 3

CSP Constraint Satisfaction Problem. 15

WSC Web Service Composition. 16

1 Introduction

1.1 Motivation

Web services have now become ubiquitous in our lives, helping us with menial tasks on a day-to-day basis, like looking for a flight, asking Siri to set a timer etc. Unfortunately, web assistants are currently incapable of executing complicated tasks which cannot be properly answered through a simple question or query. The web is saturated with a lot of individual web services and assistants incapable of communicating with each other. Thus, the need for a more domain-independent web service, capable of connecting to several other web-services becomes more pressing as our requests grow more complicated. We propose a solution with a specific domain to tackle the possibility of web service compositions. By incorporating temporal and spatial models, we manage to provide results to the user much faster and easier, than using different web services for every task. By evaluating the run-time and the expectations of users, we compare our solution to the conventional methods of executing certain tasks. For example, if one decides to travel, currently, one would need to go through several websites to compare rates and fees before ultimately settling on what they perceive to be the best possible option. However, information is often overlooked, such as national holidays, visa requirements etc., thus widening the margin for error when travelling. [RSe03] This gives us the impression, that a new way of booking for travelling is needed and the solution presented in this thesis can be a good alternative.

1.2 Example Scenario

Consequently, we will attempt to replicate the behavior of a person when they try to book a trip or vacation, by using only a single web service capable of communicating with other web-services, thus simplifying the process for the end-user exponentially, similarly to this paper [Eir16]. By utilizing the constraint satisfaction problem approach [KLA09], every particular concern can be accounted for, such as – budget for flights and hotels in a certain area and weather conditions. Whether or not every requirement has been met can be reviewed and compared, in order to achieve an optimal set of possible solutions, or fail to find one if the requirements prove impossible to satisfy. Every requirement can be mapped to an Application Programming Interface (API) of an already existing service,as is described in [Ale11] and [J F05] , for example, when searching for a hotel nearby,the existing API of Booking.com or Airbnb can be used, for the flight search can be used skyscanner, for weather check can be used openweathermap with up to 30-day forecast. By compiling all available information, all specified requirements could potentially be satisfied and presented to the user via an intuitive easy-to-understand GUI or Web App, allowing the user to make an informed decision without the need for visiting multiple different sites to get this information. Initial information

can be inputted by the user into the Web App or GUI and the result can subsequently be visually represented by a graph, as well as text, ensuring that every user is able receive and comprehend the given information [GZ09].

1.3 Structure of the thesis

In chapter 2 a necessary background knowledge is presented. Chapter 3 shortly gives us an overlook of the researches done in this area. Afterwards the design and idea of the problem and solution are introduced in chapter 4 and in chapter 5 our implementation and example solution is described. In chapter 6 the evaluation of the design and implementation are given and the thesis concludes in chapter 7.

2 Background Information

2.1 Mathematical algorithms

The most important approach, that we discuss is the **Constraint Satisfaction Problem (CSP)** [Sal98]. The idea of the approach is assigning some requirements or questions to sets of variables and with a predefined sets of constraints they are mapped in order to satisfy a specific number of constraints. The problem is NP-complete and it is very researched in both Artificial Intelligence and operations research.

Definition 2.1.1
The constraint satisfaction problem is a triple defined as P = (X, D, C)

- $X = X_1, ..., X_n$ *is a set of variables*
- $D = D_1, ..., D_n$ *is a respective set of domains for each variable*
- $C = C_1, ..., C_n$ *is a set of constraints*

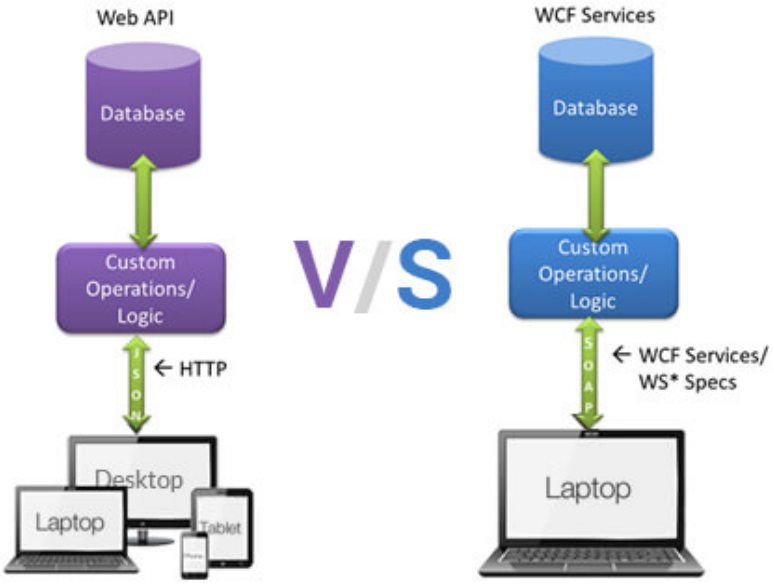
Allen's interval calculus [All] is a calculus introduced by James F. Allen for reasoning temporal models. It defines different relations between temporal intervals and also gives us a composition table, that allows us to reason of events or temporal descriptions. The high level idea of the calculus is to define the relation between two or more time intervals, for example if they are overlapping, equal, precede each other and others.

2.2 Communication methods

Web service - There is no clear definition of the term web service. A Web service can be, either communication between two computers via the internet, or a server listening to requests over the network. They provide web-based interface to the data server and exchange data between systems or applications. They are autonomous and are identified by URLs [Ion].

Web Application Programming Interface (API) is a software interface that allows two applications to interact with each other without any user intervention. APIs provides product or service to communicate with other products and services without having to know how they are implemented and it is also a tool to push data to the server that can be processed by server code or may be saved in any persistent layer.Web API works on HTTP and it can be also called or consumed by any kind of application like Mobile App, Desktop App, and Web Application etc. Thus,Developing Web API will give you a broad range of the Interface application,which can consume your Web API and fetch information to show the users. Web API is not limiting you to stick with specific interface or technology in order to interact with users.Most web services provide an API, which, with its set of

commands and functions, is used to retrieve data. Here is one example: Twitter delivers an API that authorizes a developer access tweets from a server and then collects data in JSON format. All web services can be APIs, but not all APIs can be web services.[Cle]



The most important **difference between an API and a Web service** is that web services must always be accessed through a network and an API isn't always accessible over the internet. A web service is always considered an API, but not all APIs are web services. Another important difference is that an API is platform independent. It can be accessed from a mobile app or a website and a web service is more limited. Web Service Composition (WSC) - has the task of combining already existing Web services and creating a new Web service. For example, when shopping online, we use the web service of the online shop and another web service when paying. We can use a WSC if the output parameters of one service can be used as the input parameters of another service, these two services can be connected as a new service.

3 Related Work

3.1 Domain independent planning for web services

In the paper [Mar16], the writers do a research into automated planning that provides new insights into the composition of services and helps provide automatic compositions that adapt to the changing needs of users and environmental conditions. Most of the non domain-independent solutions cannot efficiently process numerically valued variables, particularly recognition results or operator input, and do not take into account the restoration of run-time contingencies due to incorrect service behavior or exogenous events that disrupt the execution of the plan. The proposed solution uses the constraint satisfaction problem. In order to meet the requirements of the service domains, the RuGPlanner is equipped with a number of special features, including a representation of the state of knowledge for modeling the uncertainty about the initial state and the result of the detection actions as well as the efficient handling of numerically evaluated variables. In addition, it generates plans with a high degree of parallelism, supports rich declarative language to express advanced goals, and allows continuous review of the plan to handle recognition results, errors, and long responses. Times or timeouts as well as the activities of external agents. The proposed planning framework is assessed based on a number of scenarios to demonstrate its feasibility and efficiency in different planning areas and execution conditions that reflect the concerns of different service environments.

3.2 Continual planning for web service composition

Kaldeli, Lazovik and Aiello [KLA11] talk about the challenges of aggregating loosely-coupled software components with the idea to provide more functionality. By applying a framework that uses the Constraint Satisfaction problem, they try to solve the composition of web services problem and add more value-based functionalities to the solution. They propose an algorithm for interleaving planning using and changing the constraint satisfaction problem using the feedback provided from the run-time. They prove that their solution is better than previous approaches with demonstrating that the product can be used in various situations and evaluating scenarios using real web services.

3.3 Web service composition

In this work [Mar09], the main focus is the automatic web service composition and its problem of the on-demand combination of loosely coupled service operations, which serve the purpose to realise some complex task for the user. They present an approach that deals with the problem of service composition using the constraint satisfaction problem. The writers also introduce a language

that can express extended goals, when it is quipped with temporal constructs and maintainability properties to avoid undesirable situations. Using the constraint satisfaction problem, the solution proposed can model domains and goal via constraints in order to compute a valid plan.

4 Design

The Service composition we propose is not just an aggregate of different services, but more appropriately described as a model [CST12]. A composition of different services is very complicated, because every service has a different input and response. For example, some services use the name of the destination city, others make use of a special code saved on their database. Additionally, coordinates can also be used for this purpose, which are very exact. A composition between services with different spatial or temporal models [BP03] is particularly difficult and we need to examine the different models closer.

4.1 Architecture

Before we analyze the different models we need to define a general architectural design. The Web Services architecture describes the way to instantiate the elements and implement the operations in a practical manner.[Poi]

The architecture of a web service consists of three roles: service provider, service requester, and service registry. The service provider hosts a network-associable module, a web service and someone posts it to a service requester or service registry. This service requester uses a lookup operation to get the description of the service locally or from the service registry. [Mic05]

The high level idea and general architecture of our web service composition is, a user uses the composition to find their next trip. After giving all the information needed, our web service rewrites the data in search queries and using API calls to other services, the information for the end result is obtained. After careful processing of the data received and composition of the responses, the web service delivers final set of solution or solutions. They are appropriate to the information the user has provided and the conditions set. [Got02]

4.2 Temporal Model

The following section will serve to define what a temporal model is, as well as to provide context regarding its relevance towards providing a solution to the service composition problem. When using different services,an important aspect which plays a large role is what information they require and provide us. Simply aggregating the services is not a valid solution and it will result in errors and confusing output data.

For example - if an individual from Europe were to plan a vacation to Australia. We can assume that this trip will last from the 1st till the 10th of any given month. A traditional service or assistant would book a flight as well as a hotel for the relevant dates. However, a flight from Europe to

Australia would take 22 to 40 hours, depending on connections and delays. Thus booking a hotel on the 1st is superfluous as the guest will still be in transit during this time. This redundancy can be avoided by incorporating departure and travelling time into the CSP.

Definition 4.2.1
We know that the constraint satisfaction problem [Mar06] is a triple defined as P = (X, D, C)

- *First the variables must be defined*
  - *departure time as V1*
  - *traveling time as V2(not only flight duration but also approximate transportation time to the hotel)*
  - *It is imperative also to define the hotel check-in as V3.*
- *The domains of all variables, that we defined V1, V2, V3 are all the same [0-24], corresponding to the times in the day.*
- *Subsequently, we must define the constraints*
  - *First we have to define* $V1 + V2 \geq V3$*, which implicates that we have to arrive at the hotel at the check-in time or later*
  - *we also define V1+V2<V3+24h*

Maintaining the above-given example. If the departure time(V1) is on the 1st at 14 o'clock and traveling time (V2) is 40 hours, then arrival at the hotel must occur on the 3rd at 6 o'clock. If the hotel is booked on the 1st and the check-in time is at 14 o'clock the last constraint is not fulfilled, which will mean the hotel has been overbooked. In this case the web service considers booking the hotel on the second, which will satisfy both constraint and there will be no overbooking.

Furthermore, time zones are an additional point of contention. If different time zones are not appropriately implemented in any travel solution, then this is conducive to a wide assortment of potential issues such as missing a flight, not having accommodation booked upon arrival or, as in the example, overbooking accommodation. This is why, the different services should be connected perfectly, so that there are no mistakes of that nature and also the initial input should make sense to them and converted in the right value if needed. To avoid that we need to rework our P definition

Definition 4.2.2
We know that the constraint satisfaction problem is a triple defined as P = (X, D, C)

- *First we define all variables*
  - *departure time as V1*
  - *traveling time as V2(not only flight duration but also approximate transportation time to the hotel)*
  - *The important part here is also to define the hotel check-in as V3.*
  - ***We need to define a forth variable V4, which will represent the time difference between the two locations***
- *The domains of all variables, that we defined*

  - *V1, V2, V3 are all the same [0-24], corresponding to the times in the day.*
  - ***The domain of the value V4 should not only be positive, but also should be restricted to 12 to avoid mistakes.***

- *Then we define the constraints*
  - *First we have to define* $1 + V2 \geq V3$*, which implicates that we have to arrive at the hotel at the check-in time or later*
  - ***After incorporating the value in the constraints above, we can notice a serious change in the second example. The new constraint should look like this V1+V2+V4<V3+24h.***

If we consider the same times as in the example before, but also add the time difference of 8 hours, we will notice that we arrive on the 3rd at 14 o'clock, which will not satisfy the last constraint and the web service has overbooked. Of course, the web service should change the date of the hotel to avoid that.

There are many different ways that a time can be represented. It can be in following forms:

- dd.mm.yyyy - representing the date (d - day, m - month, y - year)
- CW - representing the week of the year (CW- calendar week)
- MM - representing the month (M - month)
- HH:MM - representing the time (H - hour, M - minute)

The representation should be intuitive to the users, but also for the different services. As discussed before, different services may use different time models, this is why the proposed solution is to pick one representation model that will be used not only for the input, but also for the result and if needed this value can be transformed or converted. Additionally for time values, it is not so complicated to change the format or just to take the week, in which the date is, but for spatial model it can be quite difficult.

4.3 Spatial Model

Which brings us to the next important model, the spatial model. The same goes for the location input here as well. If the flight service needs a name for the destination and the hotel service needs coordinates, the initial input that the user gives should be converted for one of the services in order to receive an appropriate result. There are a lot of ways to represent a location. As mentioned before that could be one of the following

- name of a city
- ZIP code
- coordinates
- specific code, created from the service

This can further complicate the solution. For most of the options, there shouldn't be only the input and some conversions, but also a different call should be made. Let us take an example.

If we use flight and hotel service that use different spatial models, let us assume the flight service uses the name of the destination city and the hotel service uses coordinates of the place, one will have to pick only one of the representations for the place to use for the input and results, because otherwise there will be confusion. It should not be expected from the user to input a name of the destination and as a result to receive coordinates or a code as a result and vice versa.

For the current example it will make more sense to use the name of the destination city, since there aren't a lot of people that know the coordinates of every city and also if the user doesn't know them, they will have to search for them, which defeats the purpose of this solution, since the goal is to make the search much easier and quicker. That means the appropriate solution will be to pick the most common representation, meaning name of the city.

After the composition receives the information, it will be easy to find the flight, but it wouldn't be possible to use the hotel service since it requires a different input and an answer wouldn't be received. A conversion in the background should happen and, in this scenario, a different service should be used in order to obtain coordinates of a location. This will make the receiving of the results a little bit slower, since we have to make another call, but a communication between the services will be made and there could be results presented to the user.

The same goes if the hotel service used a ZIP code or code from their database, another call should be made in order to complete the conversion of the location value. The best way to solve this solution is to look at what inputs the different services require. After taking the name of the destination from the input of the user, we can make as much transformations in the background, without requiring the users input anymore.

Before starting the search, we can ask the services, what kind of inputs they require and prepare the variables accordingly.

Definition 4.3.1
We know that the constraint satisfaction problem is a triple defined as P = (X, D, C)

- *First we define all variables*
  - *initial input as variable* V_0
  - *for every required input from the services we define* $V_{1..n}$*, where n represents the number of web services requiring that input*
- *The domains of all variables* $D_{0...n}$*, will be the same as defined from the corresponding web services, so that the structure can be the same.*
- *And the only constraint here will be, if the output* V_m *is different from* V_0*,* V_m *needs to be transformed to match it, where m is the output of the last service, that used the variable and* $0 \leqslant m \leqslant n$

After we have all variables transformed and defined correctly, we can begin with the searching with each web service and as the constraint states we can transform the output back to the initial type, if needed so and deliver it to the user, so there is no confusion.

4.4 Special case

A very interesting example is, if a hotel cannot be found, for the dates given. Most, if not all the web services will return an error.

Figure 4.1: Allen's interval calculus relation

We propose a solution using the Allen's interval calculus [KNA99]. The Relation we see in Figure 4.1, can be used when the scenario, we described occurs.

Let us explain this, considering the following example. If a user wants to book a hotel for 10 days during a national holiday, Christmas for example, when the availability is low, our web service shouldn't just return an error. Instead, it can use the Allen's interval calculus to split the stay in two or more parts, so that they don't overlap and there are no days without a hotel left. If we just split the 10 days in two, we have two stays, with 5 days each, that we can search for. If we find a solution, this can be presented to the user as a possibility. If the conditions are not met, we can increment the first part and decrement the second and do another search. This can be done respectively for each part, until the conditions are met and a hotel is found, or until there aren't any solutions found. The response time will be much higher in this situations, but the chances of finding a solution are also much greater.

Algorithm 4.1 An algorithm using the Allen's interval calculus

```
Require: startDate, endDate
1:  var duration = endDate - startDate
2:  var splitRation = duration/2 + duration mod 2
3:  if (searchForHotel(startDate, startDate + splitRation) AND searchForHotel(startDate + splitRation, endDate)) then
4:      return true
5:  end if
6:  for i=1, i<splitRation, i++ do
7:      if (searchForHotel(startDate, startDate + splitRation + i) AND searchForHotel(startDate + splitRation + i, endDate)) then
8:          return true
9:      else if (searchForHotel(startDate, startDate + splitRation - i) AND searchForHotel(startDate + splitRation - i, endDate)) then
10:         return true
11:     end if
12: end for
13: return false
```

In Algorithm 4.1 is represented how the Allen's interval calculus can be used for this problem. First we require the start date and the end date of the stay. After that we calculate how long the duration is and we split the duration in half. If the duration is an even number, we will just divide it by two, but if the duration is an odd number the modulo calculation will make sure there are no missed dates, when booking the hotel.

In line 3, we check if we can find a solution with the initial splitting of the duration of the stay. We use the method searchForHotel(), which has inputs the start date and the end date. The Method returns true, if a hotel is found and saves the results in a buffer for later processing, or it returns false if it doesn't find anything. This part is out of the loop, because there is only one case to consider. If the algorithm doesn't find any results, the for loop will be triggered, where we respectively add or subtract the number i to each stay. This way we can achieve the maximum possible options of finding a solution. If we find a solution in the mean time, the method returns true and stops searching, if this is not the case the algorithm will just return false.

4.5 Composition of the Results

After we are done with all the searching and sending of requests, we will have a lot of unprocessed data, from which we need to extract the most valuable and meaningful information [Inc07]. Furthermore we need to deliver the best possible solution/solutions to the user.

Depending on the conditions, that the user has set, we can filter and order the results for each service. For example, if the user is not concerned about the price of the flight, we can present the shortest or one with less stops. The same goes for the hotel search, instead of showing the cheapest option, we can return a hotel closer to the city center and one that has a higher rating.

If there are no conditions set, a good options will be to return the most affordable solution and a solution that has a higher rating and it is not as affordable.

Because of the CSP integration when searching, all the solutions are compatible with each other, which makes the composition of the results much easier.

In algorithm 4.2 we represent an example of how we can order the results. This is just a simple solution of the problem. As input the algorithm takes the boolean variable price, when the variable is true, we return just the most affordable flight, but if price is false, we return the flight with the shortest duration. The algorithm can be slightly modified and instead of one variable for both the flight and hotel results, we can use two different. For example, if the user wants the shortest flight, but the most affordable hotel.

We begin with iterating trough the list of flights found and we pick the most suitable option, considering the variable price. There could be other factors that play a role in the picking of the final result, not only the rating as shown in the example. We do a iteration of all the list of hotels and again we choose the best option considering the boolean price.

Algorithm 4.2 An algorithm for sorting the results

```
Require: price, listOfFlights, listOfHotels
var resultFlight
var resultHotel
for i=0, i<listOfFlights.length, i++ do
    if price = true then
        if reultFlight = null then
            resultFlight = listOfFlights[i]
        else if resultFlight.price > listOfFlights[i].price then
            resultFlight = listOfFlights[i]
        end if
    else
        if reultFlight = null then
            resultFlight = listOfFlights[i]
        else if resultFlight.duration > listOfFlights[i].duration then
            resultFlight = listOfFlights[i]
        end if
    end if
end for
for i=0, i<listOfHotels.length, i++ do
    if price = true then
        if resultHotel = null then
            resultHotel = listOfHotels[i]
        else if resultHotel.price < listOfHotels[i].price then
            resultHotel = listOfHotels[i]
        end if
    else
        if reultHotel = null then
            resultHotel = listOfHotels[i]
        else if resultHotel.rating > listOfHotels[i].rating then
            resultHotel = listOfHotels[i]
        end if
    end if
end for
return resultFlight
return resultHotel
```

5 Implementation

For the implementation of the solution, we created a small website, which has an example form for the user to fill out and begin with the searching.

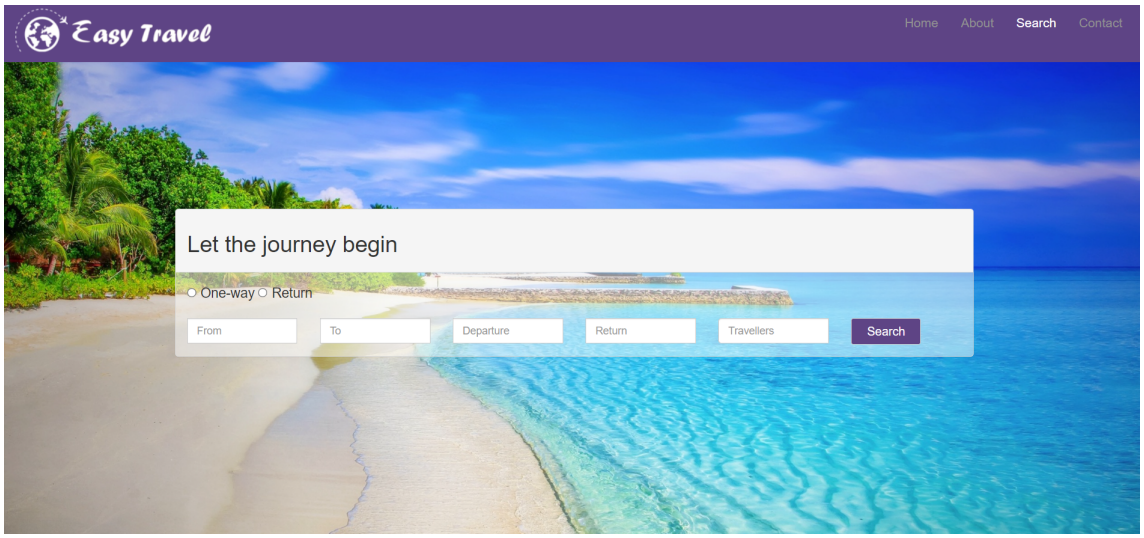


Figure 5.1: The website form created

As shown in figure 5.1 the form contain just a few inputs:

1. radio buttons if the journey is one-way or return
2. start location
3. desired location
4. departure date
5. returning date
6. number of people travelling

5.1 Used APIs

This is just the basic information that we need for the implemented example. After the user fills out the form and clicks on the search bottom, all the information is transformed into queries and the requests are send one after the other. For this we need 3 APIs:

1. Flight search API [teae]

2. hotel search API [teac]
3. Weather search API [teaf]

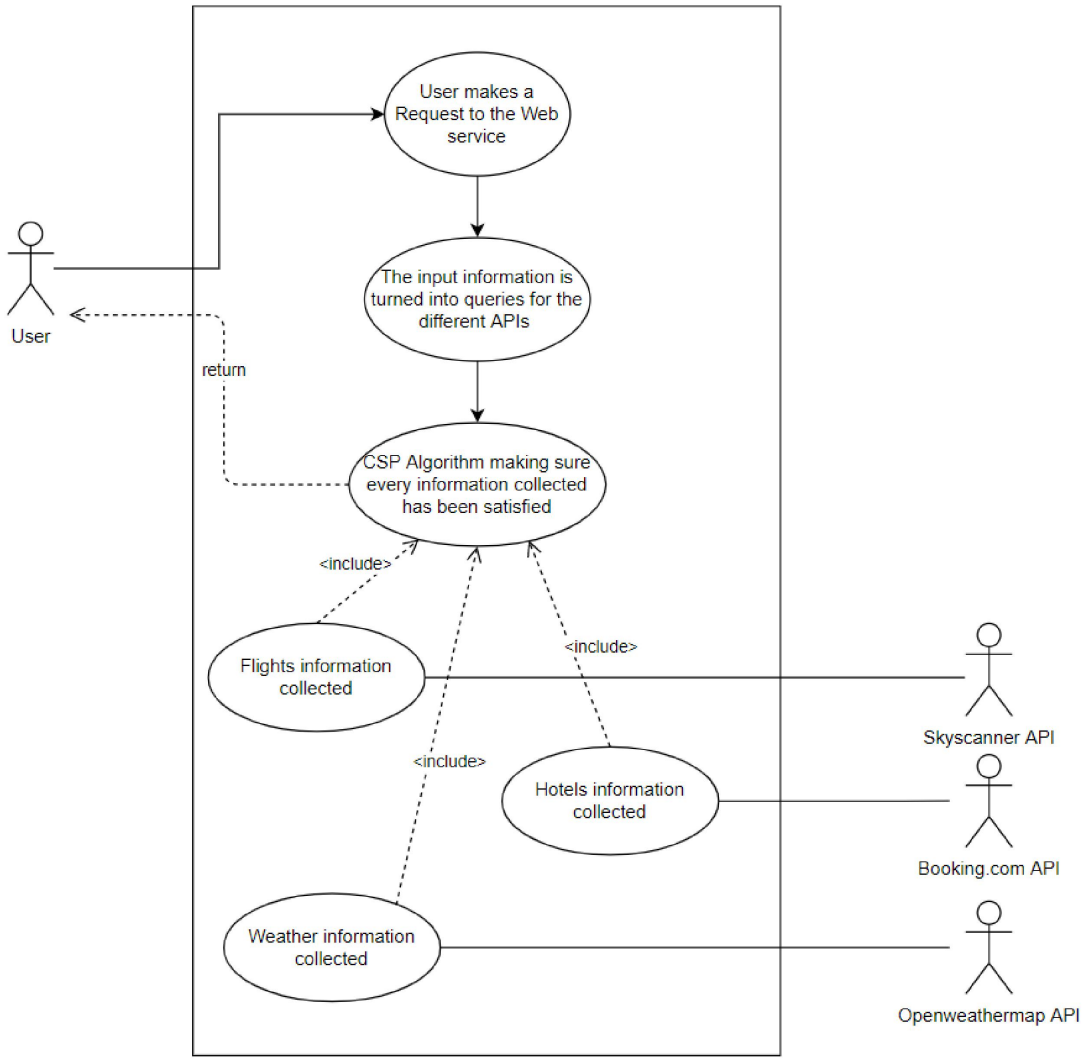


Figure 5.2: Implementation use-case

Figure 5.2 shows how we implemented our solution in the backend.

We use the rapidapi.com [tead] service, that provides all kinds of APIs. Some of them have basic free plans with limitations, for example quota of requests per month, or number of requests per second or some of them have limited access to the different calls. For the flight search we decided to use the skyscaner API, because there were no limitations for the requests sent per month. In order to find a result, a different call must be used first, that will give the code of the airport of the desired city. After that the code can be inserted as a parameter. The response gives us information about the departure date, if the flight is direct, the price and a lot of other information that may not be used. For the weather forecast we chose weatherbit, because the request quota per month was the highest and the API gives us up to 16 days(daily) forecast. In this option a different API must be used first in order to find the latitude and the longitude of the desired city. The API we chose is

GeocodeSupport [teab]. After receiving all the needed information for the location, the call can be used, and the response gives a very detailed information. This API will be used only if the user chooses a date 16 days from the current date, unfortunately we were not able to find a free API that gives a forecast for more than 16 days ahead. And lastly the hotel search API is hotel4. First a call is made to find out what is the destinationId of the desired place. After that we can receive information ordered by price or star rating about the available accommodations. In the response there is information about the rating of the hotel, the price, how far is from the city center and a lot of other useful information.

5.2 Backend Logic

When implementing the solution we used, html [HOY11], css [McF12] and javascript [Gre10] for the frontend and javascript, axios [npm], Node.js [HOY10] and ajax [Gar05]for the backend.

After receiving all the information from the frontend, we parse the information to the backend, where we make all the calls to the APIs. Initially only two calls are made, to get the airport codes from the flight search API, after that the actual flight searching is done. The next step is looking for the hotel and then finally a call to the weather API is made. All the results are passed to the frontend and presented to the user.

Algorithm 5.1 Ajax request to the backend

```
await $.ajax({
            url: 'http://localhost:3000/flight/'+from+'/'+to+'/'+output["departure"],
            dataType: 'text',
            success: function(data) {
              oneway = data;
            },
            type: 'GET'
        });
```

Algorithm 5.1 is an example request, we send to the backend. First we define the url, with the corresponding attributes(departure city, arrival city and the date of departure). This request is for an one-way flight search, this is the reason, we don't define a return date. The response data is in form of a text, because we don't return all the information the API sends us back. After that we safe the information in the variable one-way and last we define that this is a GET function.

In Algorithm 5.2 is shown, how the same request looks like in the backend. The backend is listening for the particular url we defined in the frontend and after a request is done, the following method is triggered. We first safe all the information from the request in variables. The next step is configuring the request to the API. We define, that it is a GET request, the url for the API with the corresponding variables and the parameters. The headers we use are provided from the rapidapi service, which allow us to use all the APIs used in the solution. After everything is set up the call is made using axios and the result is sent back to the frondend.

Algorithm 5.2 Request from the backend to the API

```
app.get('/flight/:FROM/:TO/:DEPARTURE', function (req, res) {
  async function example() {
    let from = req.params.FROM;
    let to = req.params.TO;
    let departure = req.params.DEPARTURE;
    try {
            var config = {
                method: 'GET',
                url: 'https://skyscanner-skyscanner-flight-search-v1.p.rapidapi.com/
apiservices/browseroutes/v1.0/US/USD/en-US/'+from+'/'+to+'/'+departure,
                params: {inboundpartialdate: departure},
                headers: {
                    'x-rapidapi-key': '6f3c39a089msha49a9b46913e4eep12aed4jsn235cf3969736',
                    'x-rapidapi-host': 'skyscanner-skyscanner-flight-search-v1.p.rapidapi.com'
                }
              };

              axios(config)
              .then(function (response) {
                //console.log(JSON.stringify(response.data));
                res.status(200);
                res.send(response.data);
              })
              .catch(function (error) {
                console.log(error);
              });

    } catch (err) {
      console.error(err);
    }
  };
```

6 Evaluation

6.1 Design of the Experiments

For the evaluation we did two different experiments with ten people. The first experiment was using the solution and the conventional services, in order to receive information about the run-time. All experiments were made on the same computer and internet connection to avoid anomalies in the results. To avoid any confusion we also had all three conventional websites open on their main page, as well as our solution. This way we eliminate the time of searching for the services and only the searching in the services and the response time is considered. The second one was series of 5 questions using google forms [teaa]:

- Do you use personal assistants?
- How often do you use personal assistants?
- Would you use a personal assistant or a composition web service to find your next trip?
- Have you used a personal assistant or a composition web service to find a trip?
- Would you trust a personal assistant to book your next trip, without manually checking the results?

Additionally we asked for the age of the subjects, to see if there is any correlation between answers/run-time and their age.

6.2 Results

As mentioned in the previous section, before asking the users to use the solution or answer any questions, we asked them about their age. As shown in figure 6.1, the participants are mostly between 18-38, since these are the users that use services with travelling domain the most, but there are representatives from all age groups.

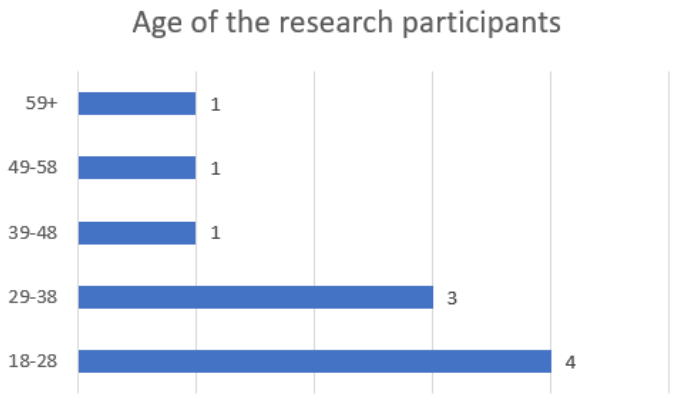
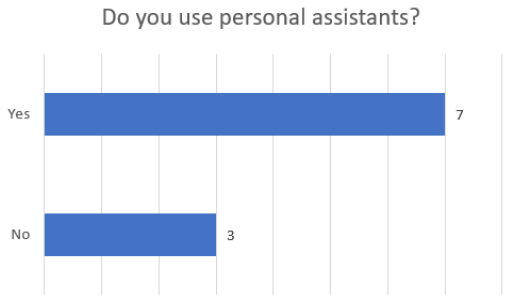
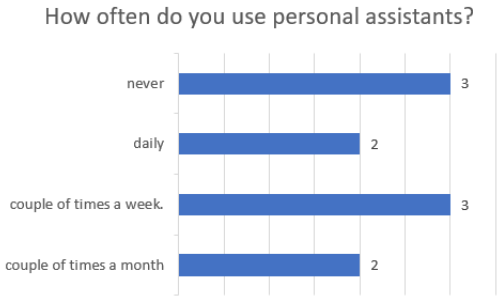


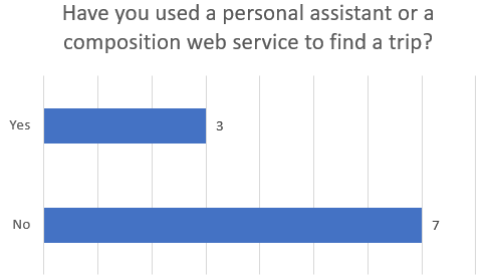
Figure 6.1: Age of the research participants

All the users were asked the same set of five questions as described in the previous section. The first two are to get an idea, if they use personal assistants and how often. Only three of the people have never used a personal assistant and they are the people from the age groups above 39. The rest of the users use personal assistants at least few times a month.

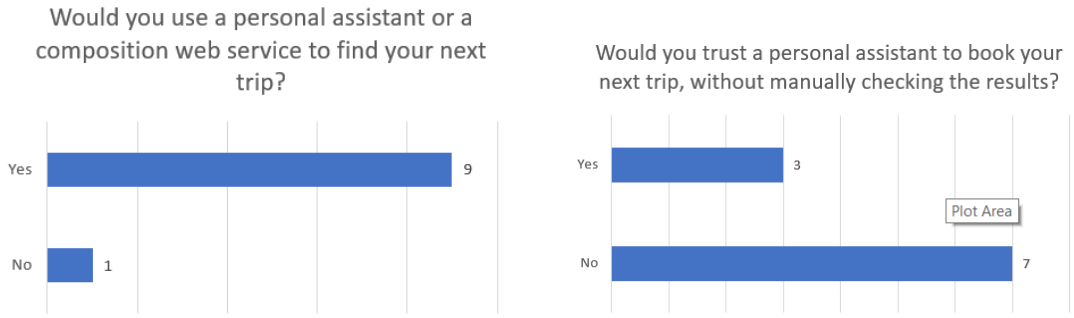




When asking the next question, if some of the users have used a web composition to find a trip, three of the participants answered with yes. When asked to give an example, they all answered skyscanner []. In our opinion the answer is valid, but this web service composes the results of another web services with the same purpose, meaning they all have the same temporal and spatial model and return results with the same structure. It is good to see that the research participants understand the meaning of the question and to see, if they actually remember that the composition possibly did better job than the conventional method of searching for flights.



The last two questions that we asked, help us to get an idea if the people who participated in the experiment want to use a personal assistant or web service composition when looking for a trip. Surprisingly nine people would use one, which gives us the information that they are ready for this next step in the development in the personal assistants and web services, but only three of them would actually trust it to book their trip.



Finally we tested the run-time of our implementation compared to popular search engines for all the services. All the users were provided with the websites already open and were given the task to find results for the same trip. A return ticket from London to Paris, seven days hotel in Paris for two people and additionally checking the weather in Paris. As it shown in figure 6.4, the run-time when using our proposed implementation is more than twice as low compared to using conventional searching tools. In the run-time evaluation the only thing considered is typing the information and waiting for a response. When using the implementation described in chapter 5, the response time is relatively high, because of the composition of the results and the fact that several different calls to different APIs are made.

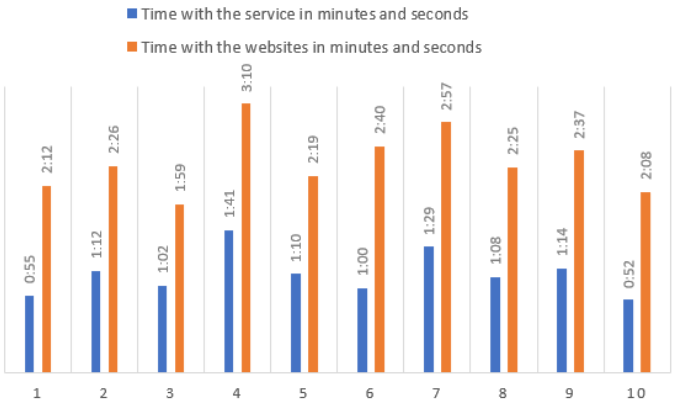


Figure 6.4: Run-time experiment results

6.3 Summary of the results

When looking at the results, we can see a trend that people from the higher age groups are unfamiliar to web services and personal assistant, but some of them are willing to use one, when looking for their next trip. Their response time, when using both the implementation and the conventional websites is also higher compared to the other participants.

Another interesting aspect of the results is that most of the participants in the lower age groups have used personal assistants and web service compositions and they are willing to use them again, but they won't trust them when booking their next trip, without manually checking the results.

7 Conclusions

7.1 Summary

Overall the conclusion we can make is, that the proposed implementation from chapter 5 provides a twice as fast solution, when looking for a trip. The initial definition of the API calls and constraints would cost some time, but when used by a lot of users, lowering the response time and effort from the user's side will be useful. Furthermore when evaluating the results, we can see that most users are ready to use such a solution.

This thesis defines an approach of composing web services by using different temporal and spatial models, to achieve perfect communication between the services. Using the CSP, we manage to define several constraint, when looking for appropriate results and eliminate unpractical solutions.

The results we received from the evaluation show us that the proposed solution is appropriate, when looking at the run-time and the research participants are willing to use it further. Perhaps after using the solution in real time and in a real situation, they could build a trust towards the assistant or the service and they could even book their trips, without manually checking the results beforehand. It is important to remind, that the web service was tested locally with real data and APIs and only providing realistic results in the console log.

7.2 Future work

The developed solution can be optimized further to lower the response time, update the interface, better the composition of the API responses and extend the functionality. In this current climate, adding the government requirement, when entering a different country can be very useful to the end user. For example, vaccination requirements, test before arrival or after arrival, quarantine obligations etc. More services can be added to extend the final information provided to the user.

When talking about the development of personal assistant, we have to consider also the domain independent services. When using them, we ask and search for different solutions every time. In most cases they just google, what we look for and give us the response. This won't be an appropriate solution, when looking for a trip. Even if we add similar solutions for a lot of different scenarios, this will not make the assistant or the service domain independent. The constantly changing APIs, will start delivering errors and we still can't cover every need of the user.

Bibliography

[] *Sky scanner*. URL: https://www.skyscanner.de/ (cit. on p. 32).

[Ale11] M. A. Alexander Lazovik Eirini Kaldeli. "Continual Planning with Sensing for Web Service Composition". In: *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence* (2011) (cit. on p. 13).

[All] J. F. Allen. *Allen's interval algebra*. URL: https://en.wikipedia.org/wiki/Allen27s_interval_algebra (cit. on p. 15).

[BP03] J. Y. Bart Orri¨ens, M. P. Papazoglou. "Model Driven Service Composition". In: *M.E. Orlowska et al. (Eds.): ICSOC* 14 (2003), pp. 75–90 (cit. on p. 19).

[Cle] T. Cleo. *What are API and Web Service*. URL: //www.cleo.com/blog/knowledge-base-web-services (cit. on p. 16).

[CST12] M. Carman, L. Serafini, P. Traverso. "Web Service Composition as Planning". In: *ITC-IRST, Via Sommarive* 18 (2012) (cit. on p. 19).

[Eir16] M. A. Eirini Kaldeli Alexander Lazovik. "Domain-independent planning for services in uncertain and dynamic environments". In: *Artificial Intelligence* 236 (2016), pp. 30–64 (cit. on p. 13).

[Gar05] J. J. Garrett. "Ajax: A New Approach to Web Applications". In: *Medical Reference Services Quarterly* (2005) (cit. on p. 29).

[Got02] K. Gottschalk. "Introduction to web services architecture". In: *Ibm Systems Journal* 40 (2002), pp. 170–179 (cit. on p. 19).

[Gre10] B. B. J. V. Gregor Richards Sylvain Lebresne. "An Analysis of the Dynamic Behavior of JavaScript Programs". In: (2010) (cit. on p. 29).

[GZ09] D. Gotz, M. X. Zhou. "Characterizing users' visual analytic activity for insight provenance". In: *Information Visualization* 8 (2009), pp. 42–55 (cit. on p. 14).

[HOY10] M. B. HOY. "Node.js: Using JavaScript to Build High-Performance Network Programs". In: *IEEE Internet Computing* 14 (2010), pp. 80–83 (cit. on p. 29).

[HOY11] M. B. HOY. "HTML5: A New Standard for the Web". In: *Medical Reference Services Quarterly* 30 (2011), pp. 50–55 (cit. on p. 29).

[Inc07] D. M. Incheon Paik. "Automatic Web Services Composition Using Combining HTN and CSP". In: *Seventh International Conference on Computer and Information Technology* 1 (2007) (cit. on p. 24).

[Ion] Ionos. *Dienste von Maschine zu Maschine*. URL: https://www.ionos.de/digitalguide/websites/web-entwicklung/webservice/ (cit. on p. 15).

[J F05] S. K. J. Fan. "A snapshot of public web services". In: *SIGMOD Rec* 34 (2005), pp. 24–32 (cit. on p. 13).

[KLA09] E. Kaldeli, A. Lazovik, M. Aiello. “Extended Goals for Composing Services”. In: *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling* (2009) (cit. on p. 13).

[KLA11] E. Kaldeli, A. Lazovik, M. Aiello. “Continual Planning with Sensing for Web Service Composition”. In: *AAAI Conference on Artificial Intelligence* 25 (2011), pp. 1198–1203 (cit. on p. 17).

[KNA99] M. KNAUFF. “The cognitive adequacy of Allen’s interval calculus for qualitative spatial representation and reasoning”. In: *Spatial Cognition and Computation 1* 1 (1999), pp. 261–290 (cit. on p. 23).

[Mar06] D. Maruyama. *A Flexible and Dynamic CSP Solver for Web Service Composition in Semantic Web Environment*. Incheon Paik, 2006. ISBN: 8780367820196 (cit. on p. 20).

[Mar09] A. L. Marco Aiello Eirini Kaldeli. “Extended Goals for Composing Services”. In: *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling* 25 (2009), pp. 362–365 (cit. on p. 17).

[Mar16] A. L. Marco Aiello Eirini Kaldeli. “Domain-independent planning for services in uncertain and dynamic environments”. In: *Artificial Intelligence* 236 (2016), pp. 30–64 (cit. on p. 17).

[McF12] D. S. McFarland. *CSS the missing manual*. The missing manual. POGUE PRESSÖ’REILLT, 2012. ISBN: 9780198520115 (cit. on p. 29).

[Mic05] M. P. S. Michael N. Huhns. “A Semantic Web Services Architecture”. In: *IEEE Computer Society* 5 (2005), pp. 72–81 (cit. on p. 19).

[npm] npm. *Axios*. URL: https://www.npmjs.com/package/axios#global-axios-defaults (cit. on p. 29).

[Poi] J. T. Point. *Architecture of Web Services*. URL: https://www.javatpoint.com/restful-web-services-architecture-of-web-services (cit. on p. 19).

[RSe03] A. R.Sethuramana Dr.T.Sasiprabha. “An Effective QoS Based Web Service Composition Algorithm for Integration of Travel Tourism Resources”. In: *International Conference on Intelligent Computing, Communication Convergence* 48 (2003), pp. 541–547 (cit. on p. 13).

[Sal98] B. M. S. Sally C. Brailsford Chris N. Potts. “Constraint satisfaction problems: Algorithms and applications”. In: *European Journal of Operational Research* (1998) (cit. on p. 15).

[teaa] google team. *Google Forms*. URL: https://www.google.com/forms/about/ (cit. on p. 31).

[teab] rapidapi team. *Forward Reverse Geocoding API Documentation*. URL: https://rapidapi.com/GeocodeSupport/api/forward-reverse-geocoding/details (cit. on p. 29).

[teac] rapidapi team. *Hotels Overview*. URL: https://rapidapi.com/apidojo/api/hotels4/details (cit. on p. 28).

[tead] rapidapi team. *RapidAPI Consumer Quick Start Guide*. URL: https://docs.rapidapi.com/docs/consumer-quick-start-guide (cit. on p. 28).

[teae] rapidapi team. *Skyscanner Flight Search Overview*. URL: https://rapidapi.com/skyscanner/api/skyscanner-flight-search/details (cit. on p. 27).

[teaf] rapidapi team. *Weather Overview*. URL: https://rapidapi.com/weatherbit/api/weather/details (cit. on p. 28).