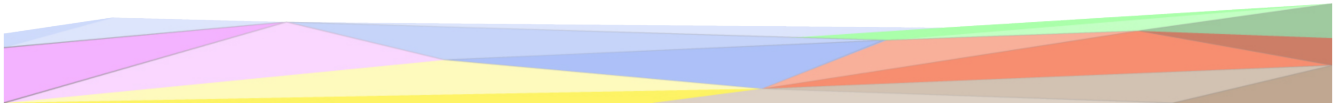
Dipartimento di Ingegneria e Scienza dellInformazione

– KnowDive Group –

KDI Report

Document Data:	Reference Persons:	
16 December 2018	203266	Giacomo Zara
	196272	Seyed Mahed Mousavi
	197812	Andrei Catalin Coman



Index:

Revision History:

Revision	Date	Author	Description of Changes
0.1	16.01.2017	Subhashis Das	Document Created
0.2	16.01.2017	Subhashis Das	Document Modified

References to existing KnowDive internal reports:

- Internal Report 1
- Internal Report 2

Abstract

This document describes the process by which we have been modeling a system meant to perform calendar integration, exploiting it in the scope of trip planning. The system interacts with the user by means of a Natural Language Processing/Generation module, and allows him to plan trips and ask for date conversion between calendars used in different countries. The novelty of our system consists of providing this back-end calendar integration, and making it accessible by anyone through a user-friendly virtual assistant, having it included in a trip planning tool. The idea is that the user will benefit from the fact of being guided by the system's interface, with particular respect to the issues that may derive from the differences between calendars. The model consists on a number of entities, among which certain defined relationships hold. The whole process is described in the sections that follow.

1 Scenario

The project consists of modeling a system meant to provide for the user the possibility of planning a journey by bus or plane, and access a smart tool of date conversion between calendars used in different countries. As for the trip planning feature, the user has to provide the desired trip date, together with origin and destination. The system will then provide a possible solution for the journey, asking questions if more information is needed. We have defined our scenarios with the purpose of identifying those real-life situations in which our system would be more likely to be useful.

1.1 Scenario 1

Francesco works in the mobility department of a very big international electricity distribution company, in the office of Siena, and his job consists of booking flights for the work-related trips of the employees: every day, he has to handle about 30 flight reservations. Francesco today has been provided with the new trip planning system, and he has just read the basic instructions on how to use it. Francesco now has to reserve a flight for Ms. Bianchi, who lives in Siena and has to go to Tehran, capital of Iran, for a meeting on Wednesday of the next week, and fly back to Italy on the day after. Francesco queries the system about flights from Rome to Tehran and back, providing Wednesday and Thursday, respectively, as departure and return days. The system provides the possible flight solutions according to the parameters. Once having identified a suitable flight, Francesco queries the system for buses from between Siena and Rome for both departure and return dates, in order for Ms. Bianchi to reach the airport, and make it back to Siena the day she comes back to Italy. This way, Francesco has identified all the suitable transportation solutions for Ms. Bianchi.

1.2 Scenario 2

Shadi has recently got married to Karam, and now she is planning their honeymoon in Venice. She needs all the related reservations, so she uses the system to check the dates. In particular, she asks what day is next *Jomeh* (Friday) in Italy, and the system informs her that it is a normal Friday. For the return, she asks about *Panj'shanbeh* (Thursday) of the following week, and the system informs her that this Thursday is June 2nd, therefore national feast in Italy. Given the consequent reduction of public services, Shadi decides to postpone the return to a couple of days later.

1.3 Scenario 3

Richard works in an International Trade Company sited in Chicago, and his job consists in hiring new staff. He needs to fly to Tehran in order to do some interviews, and he should now choose a date to travel. He uses the system in order to choose his flights, obtaining a solution which includes flying first to New York, then to Tehran. Once having chosen the flights, Richard queries the system in order to make sure of what kind of date he will be landing in Tehran. The system provides for Richard the information that the departure date for his way back to the States is on March 21st, which in Iran is the first day of the year. Therefore, Richard will move the return date to the following day.

2 Model formalization description

In order to design our model, we have started by roughly defining the set of entities and relations that would most likely be involved according to the scenarios, and then we went through an iterative process of validation and refinement. One of the first conclusions that we've drawn is that our system shall be split into two main components:

- **transportation assistant**: this component would take care of retrieving for the user information about transportation. In particular, the system can provide flight and bus solutions given origin, destination and date, which have to be stated by the user as parameters in order to query the database
- **date assistant**: this component would take care of retrieving for the user information about dates in different countries. In particular, the system can inform the user on any date with respect to different calendars, including the date being a feast or in a weekend. The user just has to provide his current country, his target country and the date he's interested in

These two components are conceptually linked to one another by the fact that the date information can be useful when it comes to planning a journey, since it may affect the choice of the trip dates.

Note 1

As far as the naming convention is concerned, we decided to adopt a variant of the *Snake case*[a] throughout our work. `This_Is_An_Example` exemplifies the convention adopted. Each word starts with a capital letter and the words are separated by the underscore.

[a] **Snake case**: `https://en.wikipedia.org/wiki/Snake_case`

For the formalization, we started by listing the simplest entities involved by our system, including their hierarchical structure:

- a super-class named `Administrative_Division`, which has two sub-classes:
  - `City`
  - `Country`
- a super-class named `Building`, which has one sub-class:
  - `Terminal`, having itself two sub-classes:
    - `Airport`
    - `Bus_Station`
- a super-class named `Event`, which has one sub-class:
  - `Transportation_Event`, having itself two sub-classes:
    - `Bus_Trip`
    - `Flight`

Among these classes, the following trivial relations hold:

- a `City` is `Part_Of` a `Country`

- a `Building` is `Located_In` a `City`
- a `Transportation_Event` takes place between two terminals of the same type, according to what specific event we're considering:
  - a `Flight` `Departs_From` and `Arrives_To` an `Airport`
  - a `Bus_Trip` `Departs_From` and `Arrives_To` a `Bus_Station`

2.1 The `Date` issue

At this point, we found ourselves facing an issue of whether to consider date (in a full time-stamp format) as data property belonging to an event or to consider it as an independent entity in our system. For that, we first searched for the state of the art solutions and tried to inherit from them.

2.1.1 Some state-of-the-art

By investigating the existing ways to address the time-zone issues, we have found that most of the solutions are not actually publicly shared. Still, we have taken into account the strategy adopted by iCalendar[1], as explained on Kanzaki[2]. This is what led us to the choice of using the UTC-calendar as a reference in order to compute date and time offsets and provide for the user a customized information according to the involved countries. An overview of the three calendars and an implementation of the conversion to Persian Calendar are described in Appendices A and B, respectively.

2.1.2 Our proposed solution

In order to model the date assistant component, we realized that date should rather be treated as an entity since we need to disambiguate between dates of different countries in order to convert them to one another. So, initially in the ER, we designed a complex entity as `date`, illustrated in Figure 1.

Figure 1: Date entity in the ER

The entity date has a `Utc_Date` as a referring date. it has many properties which we defined in order to handle the date conversions in each calendar to be modeled, (in our system Persian, Italian and American) it matches

[1] **iCalendar**: `https://icalendar.org/`
[2] **Kanzaki**: `https://www.kanzaki.com/docs/ical/`

each day of the year among American, Italian and Persian calendars, keeping track, for each day, of the month, the weekday, and whether it is a holiday or part of a weekend. This is done according to all three calendars. Having the date entity in our ER, we then took it to the ontology. Therefore, we have decided to include in our model a new entity:

- `Abstract_Entity`, having a sub-class:
  - `Date`

The choice of introducing the super-class `Abstract_Entity` was driven by the necessity of mapping our own ontology to the high-level one provided by the supervisors, as we will better explain in Section 4. Therefore, a `Transportation_Event` takes place on two dates of the same type, according to what specific event were considering:

- a `Flight` or `Bus_Trip` `Departs_On` and `Arrives_On` a `Date`

3 Lexical information upload description

According to the task requirements, we have decided to enrich our model by providing lexical information for each component of our system. In particular, as shown in Figure 2, we have provided two main types of lexical enrichment:

- **WordNet definition**: each component of the system is provided with a comment that contains the definition of said component according to the WordNet[a] [5] lexical database

- **linguistic translation**: each component of the system is provided with the corresponding translation in each of the languages spoken in the countries involved by our system (i.e. English, Italian and Fārsī)

[a] **WordNet**: `https://wordnet.princeton.edu/`

Figure 2: `Administrative_Division` lexical enrichment

3.1 Natural Language Processing/Generation

As previously stated, the user interacts with the system by means of a Natural Language Processing/Generation module. Figure 3 shows the high-level module functionality.

The interaction takes place through the initiative of the user, who expresses a request by voice. The system converts its voice into text using an Automatic Speech Recognition (ASR) sub-module. Then there are two phases, namely the recognition of the user's intent and the extraction of the information needed to execute the query. The intent classification phase can be seen as a multi-class classification task in which the text is mapped to its corresponding intent taken from a previously defined finite set of intents. However, for a classifier such as a Neural Network one based on LSTMs [1], it is necessary to convert the text into a feature vector. This can be done by converting each word into the corresponding n-dimensional representation by means of pre-trained Word

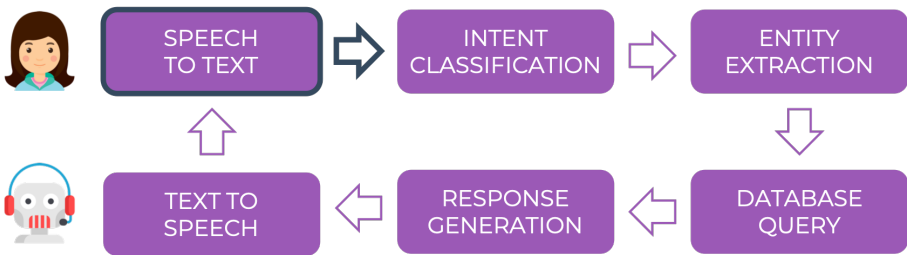


Figure 3: Natural Language Processing/Generation Module

Embeddings[3] such as word2vec [4], GloVe [6] or fastText [2]. This allows us to preserve the semantic similarity of words in a finite vector space. The hidden state of the LSTM after word-by-word processing is finally used for the classification of the intent. As far as the extraction of the entities is concerned, it is possible to use the Conditional Random Fields [3], an algorithm that has proved very effective in this task. This phase requires a dataset of sentences in which the entities that interest our domain are properly annotated. Once the user's intent is understood and the necessary entities extracted, it is possible to map this information in a corresponding query to be made to the database. The result is then passed to the response generation sub-module which, by means of templates, generates the response in the form of text. As a final step, the vocal synthesizing sub-module is responsible for proposing the response to the user in the vocal form. This process is oriented towards a high degree of interaction in order to complete the user's request. As a result, there may be several iterations before the goal is achieved.

4 Top-level grounding

A high-level ontology for Common Sense Knowledge (CSK) was provided for us by the task supervisors. The purpose of this sub-task consisted of mapping our own custom ontology to this high-level one, in order to fit our model into a more abstract representation of the world. In order to achieve this, we have gone through the CSK and started figuring out possible mappings from the domain of our ontology. This task, as mentioned, has been involved in the iterative process that resulted in the final version of our ontology itself, since the two components of the project are obviously related and co-dependent.

Here follows an outline of our mappings. We have included with light blue color our original entities, in orange color the original CSK entities and in green all the entities which we introduced in order to make the mapping compliant according to our logic.

Figure 4 simply shows how we have decided to map `Date` as belonging to an additional entity type, namely `Abstract_Entity`. This choice was due to the fact that, this way, our date entity would find its place within the logical subdivision that `Entity_Type` had in the CSK. Figure 5 shows how we have mapped our locations, namely `Administrative_Division` and `Building` (with the related sub-classes) to be themselves sub-classes of the original CSK entity `Location`. We have decided to define this specific sub-hierarchy in order to make our model more scalable, since we thought that `Building` and `Terminal` would be likely to be useful for further world descriptions. Figure 11, instead, shows how we simply considered `Flight` and `Bus_Trip` as being events of the special type `Transportation_Event`, therefore sub-classes of the CSK entity `Event`.

All the classes obviously maintain within their mappings the relations that hold among them. As for such

[3] **Word Embeddings**: `https://en.wikipedia.org/wiki/Word_embedding`

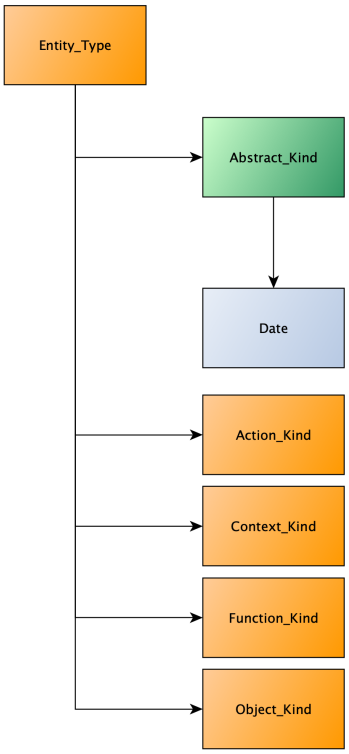


Figure 4: Date mapping

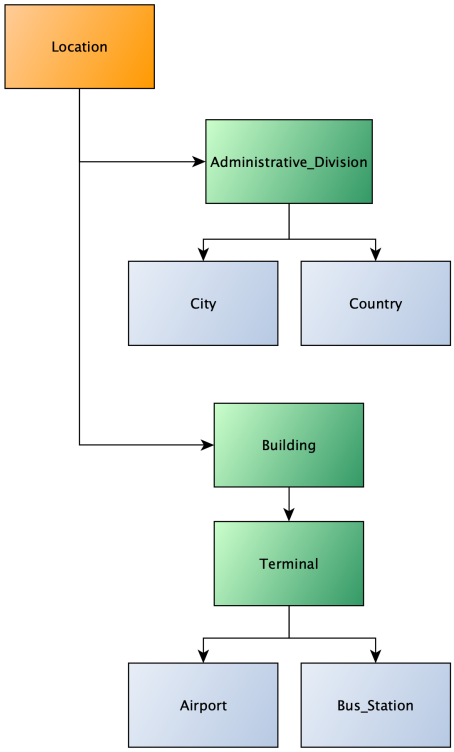


Figure 5: Location Mapping

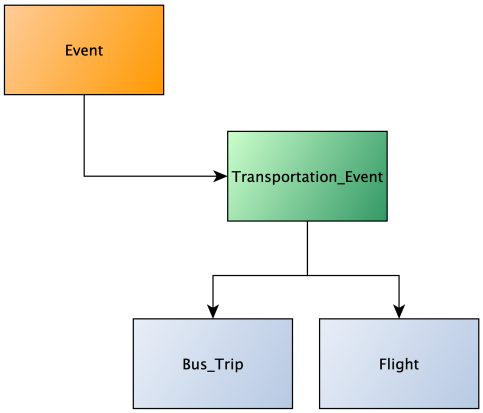


Figure 6: Event mapping

relations, we have opted for the following mappings:

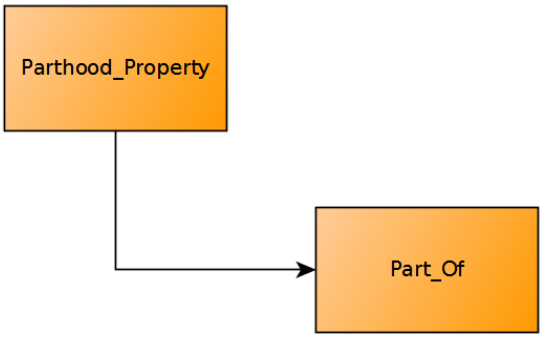


Figure 7: Part_Of mapping

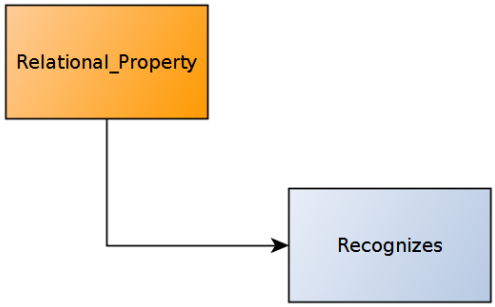


Figure 8: Recognizes Mapping

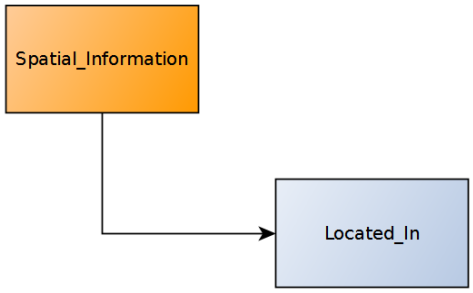


Figure 9: Located_In mapping

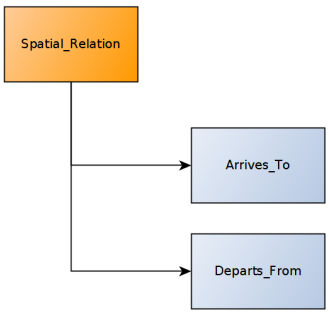


Figure 10: Departs_From & Arrives_To Mapping

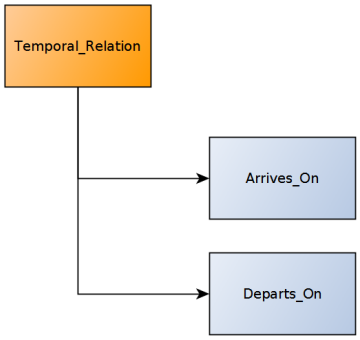


Figure 11: Departs_On & Arrives_On mapping

We have mapped our properties into being sub-properties of the items already existing in the CSK. Except for `Part_Of`, which already existed, we have followed what in our opinion was the most logical solution from a semantic point of view. Specifically, we distinguished the information domain from the relational one with respect to the travels and the location of the terminals. Finally, we thought that `Recognizes` would best fit as a relational property.

5 Model visualization

In this section, we will provide a complete visualization of our model. In order to perform the modeling, we have been using Protégé[4]. The hierarchy that holds the entities in our ontology is shown in Figure 12. As for the object properties and data properties, they are shown in Figures 13 and 14, respectively.

Figure 12:
Entities Hierarchy

Figure 13:
Object Properties Hierarchy

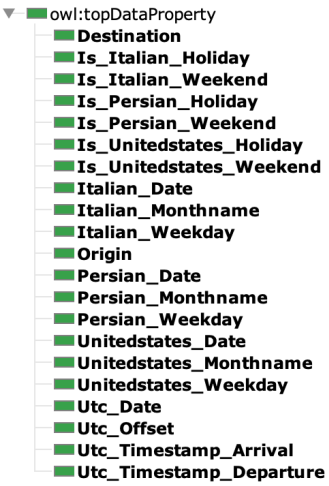


Figure 14:
Data Properties Hierarchy

For each of the entities, we have also included in the Protégé file some sample instances, on which we have based the queries described in Section 6.

Figure 15: Properties of Bus Trip

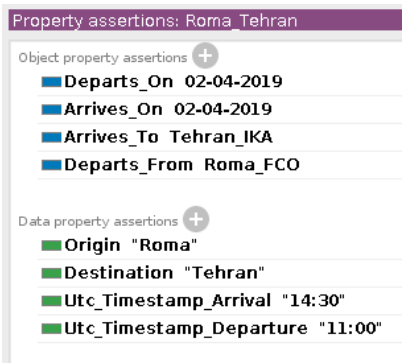


Figure 16: Properties of Flight

Figure 17: Properties of Date

[4]**Protégé**: `https://protege.stanford.edu/`

The individuals for `Bus_Trip` individual, `Flight` individual and `Date` individual are illustrated in figures 15, 16 and 17, respectively. Here we include a graphical version of the Entity-Relationship model we came up with for our system, which includes all the attributes (data properties) that we have assigned to each entity.

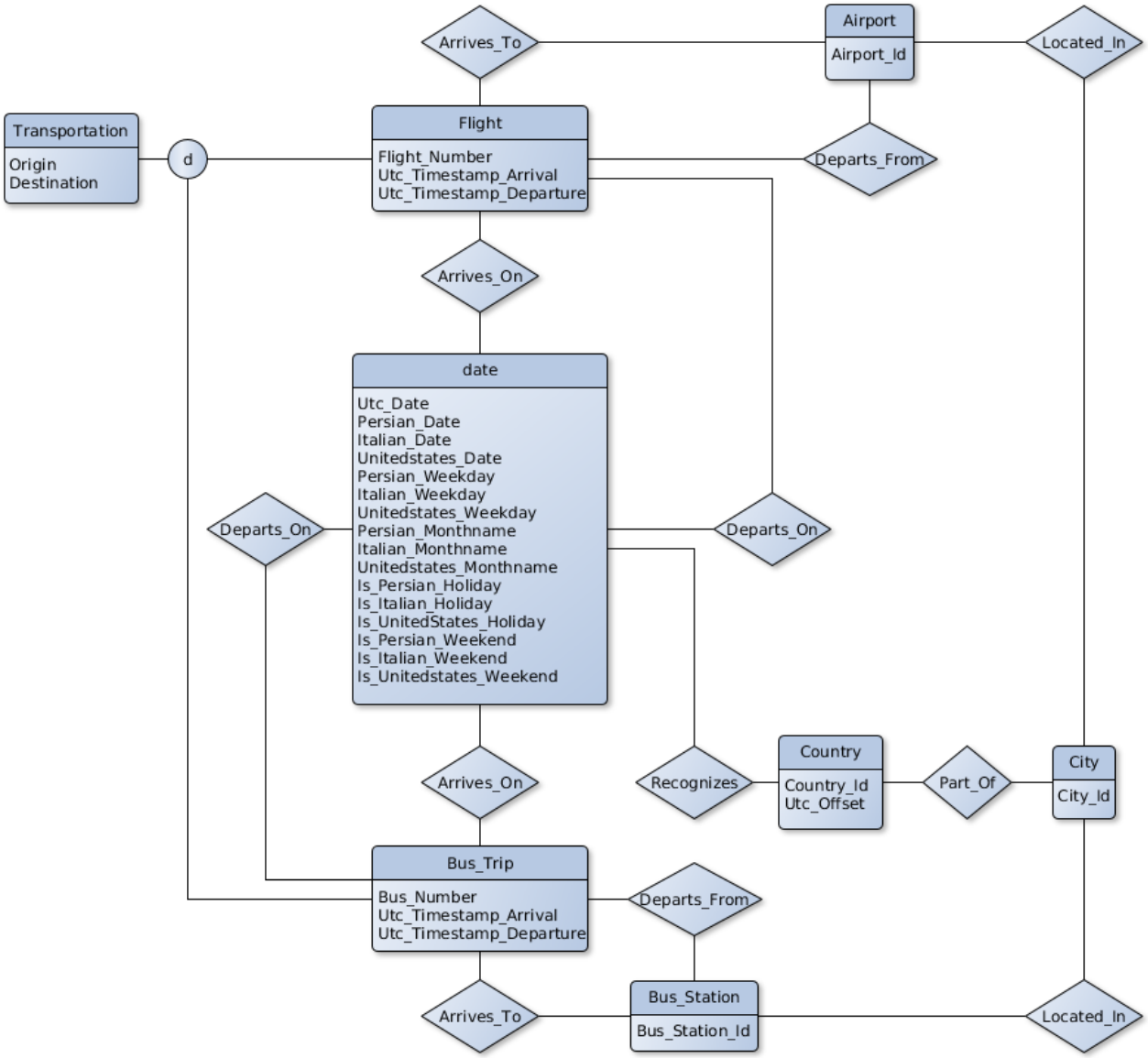


Figure 18: Entity-Relationship Model

6 Generalized queries

Our system is essentially designed in order to handle three main types of queries:

- **flights**: the user provides date, origin and destinations, and the system returns available flight solutions for the given conditions, if any exists

- **buses**: the user provides date, origin and destination, and the system returns available bus trip solutions for the given conditions, if any exists
- **date conversion**: the user provides the date according to his own country calendar and the target country, and the system provides the date according to the target country, including it being a weekend day or a national feast

In this section, we provide an example for each of these cases by means of the SPARQL Query Language[5], together with the result that the system is expected to return for each of these queries.

6.1 Flight request query

This is the query by which the user ask for flight solutions between two given places on a given date. The NLU utterance of the user would resemble the following:

"Can you tell me the flight solutions between Rome and New York on August 25th 2019?"

By means of the NLU module, the system interprets the query and formalizes it into the following SPARQL snippet:

```
SELECT ?Utc_Timestamp_Departure ?Utc_Timestamp_Arrival ?Departure
       ?Arrival ?Departure_Utc_Offset ?Arrival_Utc_Offset
WHERE {
    ?Transportation rdf:type gma:Flight .
    ?Transportation gma:Departs_From ?Departure .
    ?Transportation gma:Arrives_To ?Arrival .
    ?Transportation gma:Departs_On gma:25-08-2019 .
    ?Departure gma:Located_In gma:Roma .
    ?Departure gma:Located_In ?Departure_City .
    ?Departure_City gma:Part_Of ?Departure_Country .
    ?Departure_Country gma:Utc_Offset ?Departure_Utc_Offset .
    ?Arrival gma:Located_In gma:NewYork .
    ?Arrival gma:Located_In ?Arrival_City .
    ?Arrival_City gma:Part_Of ?Arrival_Country .
    ?Arrival_Country gma:Utc_Offset ?Arrival_Utc_Offset .
    ?Transportation gma:Utc_Timestamp_Departure ?Utc_Timestamp_Departure .
    ?Transportation gma:Utc_Timestamp_Arrival ?Utc_Timestamp_Arrival .
}
```

which would give the following result:

```
10:00, 19:45, Rome_FCO, NewYork_JFK, +1, -6
```

[5]**SPARQL**: https://www.w3.org/TR/rdf-sparql-query/

which would result in the following virtual assistant utterance (which applies the offsets to the UTC time):

"There is a flight leaving from Rome Fiumicino Airport at 10:00 (local time), arriving to New York JFK Airport at 13:45 (local time)"

6.2 Bus request query

This is the query by which the user asks for flight solutions between two given places on a given date. The NLU utterance of the user would resemble the following:

"Can you tell me the bus solutions between Isfahan and Rasht on April 2nd 2019?"

By means of the NLU module, the system interprets the query and formalizes it into the following SPARQL snippet:

```
SELECT ?Utc_Timestamp_Departure ?Utc_Timestamp_Arrival ?Departure ?Arrival
WHERE {
    ?Transportation rdf:type gma:Bus_Trip .
    ?Transportation gma:Departs_From ?Departure .
    ?Transportation gma:Arrives_To ?Arrival .
    ?Transportation gma:Departs_On gma:02-04-2019 .
    ?Departure gma:Located_In gma:Isfahan .
    ?Arrival gma:Located_In gma:Rasht .
    ?Transportation gma:Utc_Timestamp_Departure ?Utc_Timestamp_Departure .
    ?Transportation gma:Utc_Timestamp_Arrival ?Utc_Timestamp_Arrival .
}
```

which would give the following result:

```
06:00, 13:00, Isfahan_Center_BS, Rasht_BS
```

which would result in the following virtual assistant utterance:

"There is a bus leaving from Isfahan Center Bus Station at 06:00, arriving to Rasht Bus Station at 13:00"

6.3 Date conversion query

This is the query by which the user asks for a date conversion, providing the date according to his own calendar. His NLU utterance would resemble the following:

"What day is Italian April 2nd 2019 in Iran?"

By means fo the NLU module, the system interprets the query and formalizes it into the following SPARQL snippet:

```
SELECT ?Persian_Date ?Persian_Weekday ?Persian_Monthname
       ?Is_Persian_Weekend ?Is_Persian_Holiyday
WHERE {
    ?Date gma:Italian_Date ?Italian_Date.
    ?Date gma:Persian_Date ?Persian_Date .
    ?Date gma:Persian_Weekday ?Persian_Weekday .
    ?Date gma:Persian_Monthname ?Persian_Monthname .
    ?Date gma:Is_Persian_Weekend ?Is_Persian_Weekend .
    ?Date gma:Is_Persian_Holiyday ?Is_Persian_Holiyday .
    FILTER(STR(?Italian_Date) = "02-04-2019")
}
```

which would give the following result:

```
01-13-1398, Se'shanbeh, Farvardin, 0, 1
```

Which would result in the following virtual assistant utterance:

"April 2nd 2019 in Iran is Se'shanbeh 13 of Farvardin 1398, which is National Feast"

7 Evaluation of the model

	Class Coverage	Class flexibility	Attribute Coverage	Attribute Flexibility
CQ-Model	$\simeq 0.8$ (Ideal) 1	$\simeq 0.2$ (Ideal) 0.25	$\simeq 0.8$ (Ideal) 1	$\simeq 0.5$ (Ideal) 0.57
DS-Model	$\simeq 0.8$ (Ideal) 1	$\simeq 0.2$ (Ideal) 0	$\simeq 0.8$ (Ideal) 0.88	$\simeq 0.5$ (Ideal) 1
CQ-DE	$\simeq 0.8$ (Ideal) 1	$\simeq 0.2$ (Ideal) 0.25	$\simeq 0.8$ (Ideal) 1	$\simeq 0.5$ (Ideal) 0.57

Schema Level

- Does the model including cycles in the class hierarchy ? No
- Does the Model uses any polysemous terms for its class or property name? Yes
- Is Multiple Domain / Range defined for any property ? No
- Does any class have more than one direct parent class ? No
- Does the Model include multiple classes which have same meaning ? No
- Is the class Hierarchy over specified? No
- Does the model use isA as a object Property or relation? No
- Does the model have any leaf class for which there is no relation with the rest of the model? No

- Did you use miscellaneous or others as one of the class name? No
- Does the model have any chain of Inheritance in class hierarchy? Yes
- Do all properties have explicit domain and range declarations? No
- Does the model have any classes or properties which are not used? Yes
- Are a collection of elements included as a group in a number of class/attribute ? No

Linguistic Level

- Does all elements of the model (i.e. class and property) have human readable annotations? Yes
- Do all elements of the model follow the same naming convention? Yes

Metadata Level

- Is provenance information (Creator, Version, Date) available for the final protege model? Yes
- Is provenance information available for any property or class which is taken from some reference standard or ontology? Yes

8 Final considerations and open issues

We tried to do a formal modeling of calendar integration issue in this report. To this purpose, we took advantage of the transportation infrastructure as a means of showing the dates and time, which are components of the calendars, as attributes of the transportation events. We decided to model each date in the calendar as an entity which gave us considerable advantages over modeling it simply as an attribute. The advantages consist of the ability to convert one date instance of a calendar to the date instance of another calendar and covering national holidays and weekends in specific country recognizing each calendar. We then used this model as a back-end component of a trip planning system which interacts with the user through a voice user interface and provides him with possible solutions of bus-trip from one city to another city in one country in addition to international flights from the origin city to the destination located in differing countries.

In order to have a common, abstract hierarchy governing and covering all possible formal models, we mapped our ontology to the Common Sence Knowledge (CSK) and tried to place our entities and properties in the most appropriate place. It is worth mentioning that calendar integration is not a fixed problem with a fixed answer. One may address this problem differently from how other would. However, we did our best to deliver a high quality solution to this problem. Despite our efforts to solve the problem the best way, possible improvements can be done to our system. Considering the fact that date entity is very informative and useful in our system, adding all the days of the year and trying to cover as many countries as possible will result to having a huge entity in the ontology which at a certain point will be hard to handle and prone to errors. One could consider a possible optimization to our model, such as merging similar dates or excluding similar countries and improvements to our system. Besides, as mentioned in the appendix, lunar calendars are completely different from solar calendars and have their own difficulties and problems to take care of. A certain extension to our system could be merging it with lunar calendars to be used for Arabic countries.

9 Appendices

A Calendars

There are two main ways used in the world to create and formalize a calendar. The first way, used in Arabic countries, is to study and characterize the orbital movements of the moon (known as moon phases). The lunar calendars will have a year of 354 days and 12 months of 29 or 30 days each varying every month based on the visibility of the moon from a reference location. The other way, however, which is more popular, is to characterize the movements of the sun and its position with respect to reference stars. The solar calendars have 12 months and 365 days in a year. In our systems, the calendars we try to integrate are solar calendars. UTC time reference, Italian calendar, and United States calendar are all instances of a solar calendar system, known as a Gregorian calendar. Designed by Pope Gregory XIII in 1582, the Gregorian calendar is currently the most widely used calendar in the world. In this calendar, the first day of the year is the first of January which used to be the first day the consuls will enter the office and start their consular year. Below, we illustrate the two Gregorian calendars we are dealing with in our system, namely Italian and the United States, with their holidays and feasts.

Month	**# of days**	**Italian Feasts**	**American Feasts**	**Season**
January	31	1st: New Year 6th: Epiphany	1st New Year 3rd Monday: Martin Luther King Jr. Day	Winter
February	28/29	-	3rd Monday: Presidents' Day	Winter
March	31	Easter (varies) Angel Monday (varies)	-	Spring (21st)
April	30	Easter (varies) Angel Monday (varies) 25th: Liberation Day	-	Spring
May	31	1st: Workers Day	Last Monday: Memorial Day	Spring
June	30	2nd: Feast of the Republic		Summer 21st
July	31	-	4th: Independence Day	Summer
August	31	15th: Ferragosto	-	Summer
September	30	-	1st Monday: Labor Day	Autumn 23rd
October	31	-	12th: Columbus Day	Autumn
November	30	1st: All Saints' Day	11th: Veterans Day 12th: Veterans Day Observed 4th Thursday: Thanksgiving day	Autumn
December	31	8th: Immaculate Conception 25th: Christmas 26th: St. Stephen's Day	25th: Christmas	Winter 21st

Another solar calendar that we use in our system, is the Persian calendar. it is designed by Omar Khayyam, a Persian mathematician, astronomer, and poet, in imperial observatory during the Persian empire. This calendar is currently used in Persian speaking countries such as Iran, Afghanistan, and Tajikistan. The first day of the year is computed from vernal equinox which occurs in different hours of 21 of March in each year. The name of the months

are in ancient Persian, each representing a positive adjective for a month in Zoroastrian religion (Aban means the "Generosity of Sky" and Ordibehest means "Paradise on Earth") The months, feasts and holidays in Ordibehest are as follows:

Month	# of days	Feasts	Season
Farvardin	31	1st, 2nd, 3rd, 4th: New Year 12th: Republic Day 13th: Nature Day	Spring
Ordibehesht	31	-	Spring
Khordad	31	-	Spring
Tir	31	-	Summer
Mordad	31	-	Summer
Shahrivar	31	-	Summer
Mehr	30	-	Autumn
Aban	30	-	Autumn
Azar	30	-	Autumn
Dey	30	-	Winter
Bahman	30	22nd: Revolution Day	Winter
Esfand	29/30	15th: Tree Planting Day 29th Independence Day	Winter

B Date conversion algorithm

One of the commonly used methods to convert Persian date to Gregorian date and vice-e-versa, which is used by many systems, is to do few numbers of mathematics addition and subtraction of numbers and offsets based on certain conditions

For converting the Gregorian year to Persian year:
If the date occurs between 21 March and the last day of December subtract 621 from the year, otherwise subtract 622.
For converting the Persian year to Gregorian year:
If the date occurs between 1 Farvardin and the tenth day of Day add 621 to the year, otherwise add 622.
For converting the Gregorian month and day, to Persian equivalent, what these systems do is to convert the date to the number of the days passed in Gregorian year (as 22 of June is the 173rd day of the year), if the date occurs between 21 March and the last day of December, subtract 79 (as by 20th of March 79 days of passed from the Gregorian year) and otherwise, add 286 to obtain the number of the days passed in Persian year. Finally, they obtain the number of the month and day accordingly.
For the reverse, the same policy applies in an inverse way. So, 22-June-1995 is going to be:
1995-621=1374.
22 June is 173 day so 173-79=94.
which the 94th day in Persian year is 1st of Tir.
therefore, the Persian equivalent is 1-Tir-1374.

References

[1] Sepp Hochreiter and Jrgen Schmidhuber, *Long short-term memory*, Neural computation **9** (1997), 1735–80.

[2] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov, *Bag of tricks for efficient text classification*, CoRR **abs/1607.01759** (2016).

[3] John Lafferty, Andrew McCallum, and Fernando CN Pereira, *Conditional random fields: Probabilistic models for segmenting and labeling sequence data*, (2001).

[4] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean, *Distributed representations of words and phrases and their compositionality*, Advances in Neural Information Processing Systems 26 (C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, eds.), Curran Associates, Inc., 2013, pp. 3111–3119.

[5] George A. Miller, *Wordnet: A lexical database for english*, Commun. ACM **38** (1995), no. 11, 39–41.

[6] Jeffrey Pennington, Richard Socher, and Christopher D Manning, *Glove: Global vectors for word representation.*, EMNLP, vol. 14, 2014, pp. 1532–1543.