An Ontology of Emoji Characters for use on the Semantic Web

Petar Stefanov

Knowledge-based Industrial Systems Project
M.Sc. Advanced Software Engineering

University College of Dublin
School of Computer Science
Ireland

Lecturer: Dr. Tony Veale
Date: May 4, 2016

Contents

Chapter 1

Introduction

1.1 Ontology

Ontology is the study about the kind of things that exist in the universe. The word ontology comes from the Greek words - *onto* (being) and *logia* (written or spoken discourse). In philosophy, ontology is the study of what exists, the study of first principles or the essence of things. Ontologies are used to capture knowledge about some domain of interest by describing concepts from that domain and the relationships that hold between them.
In information technology, an ontology is the working model of entities and interactions in a particular domain of knowledge. In artificial intelligence (AI), an ontology is, according to Tom Gruber, an AI specialist at Stanford University, "the specification of conceptualizations, used to help programs and humans share knowledge." To share and communicate knowledge, it is important to have a common vocabulary and an agreement on the meaning for that vocabulary. Having a conceptualization is basically a mapping between symbols used in the computer and the individuals and relations in the world. Ontology provides abstraction of the world and notation for that abstraction.[3]

Different ontology languages provide different facilities. An ontology allows a pro-

grammer to specify, in an meaningful way the concepts and relationships that characterise some domain of interest.

1.2 Emoji

The word Emoji comes from Japanese *e* ("picture") and *moji*("character"). Originating on Japanese mobile phones in the late 1990s, Emoji have become increasingly popular worldwide since the boom of social networking and since they have been incorporated into Unicode, which has allowed them to be standardized across different operating systems.

Emojis in now days allows us to express the mood or feelings in digital icons, without using much text. A lot of social media users today take to a generous use of emoji text messages and at times the entire story as well.

In the social network world instead typing out the usual responses in a message communication, when something was very bad, you can just send back an emoji of a yellow face with a greenish tinge and a grimace on his face. If asked how are you? You either send back a thumbs up with a large smiling face or grumpy, crying face. Instead of using a word sentence use an emoji sentence to express yourself. It is all limited by the user imagination.

Chapter 2

ImplementationPhase

2.1 Jena and OWL

The most recent development in standard ontology languages is OWL from the World Wide Web Consortium (W3C). In general, OWL allows us to say everything that RDFS allows, and much more. A big feature of OWL is the ability to describe classes in more complex ways. For example, in OWL we can say that Plant and Animal are disjoint classes: no individual can be both a plant and an animal.

For the implementation, Jena API was used. Why Jena API? - Jena is a programming toolkit which uses the Java programming language. Jena is a Java API which can be used to create and manipulate RDF graphs. Jena has object classes to represent graphs, resources, properties and literals. The interfaces representing resources, properties and literals are called Resource, Property and Literal respectively. It allows you to work with models, RDFS and the Web Ontology Language (OWL) to add extra semantics to your RDF data.[1]

2.2 Automation

The provided spreadsheet with all unicode supported emojis - EmojiUnicodes.xlsx was carefully studied. The program was written to read and iterate through each row in the spreadsheet and build an ontology. As mentioned above, Jena Ontology API was used for building the emoji's ontology. As a starting point for classifying the possible different high level classes, the annotations from the spreadsheet were used. This website http://emojinewssearch.com/ was a good reference for how the search should be done based on search terms(Annotation) which I used to build the ontology. Decisions were made based on the most frequently used annotations. High level classes were chosen as follows:Animal(75), Person(185), Object(274), Place(148), Plant(38), Vehicle(41) and Symbol(215). All these emojis annotation names were used to build the initial hierarchy on a very high level. Basically the emojis were split into these seven main categories. Multiple iterations were done in a particular order, after the first iteration, every other iteration the check was applied to see if such instances exists and if so it wasn't manipulated. The program written iterates through the file and searches for the annotation with a name from the ones above, if found the first thing is to make this class a subclass of the high level annotation class. This approach gives us a better starting point for the manual organization of the ontology later on. During each iteration different restrictions were applied on each class.

2.2.1 Anonymous Restrictions

To define all emojis, a number of restrictions were applied on each class during each iteration. All restrictions applicable for a particular class were stored in $RDFList$. Once all possible restrictions were gathered, the final step was to apply them to the class as an anonymous restriction. An anonymous collection of all restrictions was created by using an $IntersectionClass$ which made it a super class of the current class.
The type of restrictions used was based on the annotations provided in the spreadsheet. The $EmojiDescriptor$ class was made with subclasses $EmojiColor$, $EmojiEmotion$ and

EmojiFeeling. Each of the subclasses have an object property created with a range the subclass itself and with a list of all members as an equivalent to the subclass. The *EnumeratedClass* class from the API was used to representing the list of all individuals(members) satisfying this restriction.

1. **Colour Restriction** - contains as members the annotations - Red, Green, Yellow, Blue, Black, White and Purple. Were applied 15 times.

2. **Feelings Restriction** - Feelings are senses detecting what you feel through your 11 inputs: Hearing, Taste, Sight, Smell, Heat, Cool, Pain, Pleasure, Sense of balance (vestibular),Pressure and Motion (kinesthetic). The list of members from the annotations in the file - Hear, See, Cool, Blue, Taste, Hot and Speak. Were applied 8 times.

3. **Emotion Restriction** - Emotions on the other hand are what those feelings mean. The list of members from the annotations in the file - Mad, Love, Kiss, Cry, Sad, Fear, Surprised and Disappointed. Were applied 25 times.

4. **Food Restriction** - were split in three categories: Dessert, Fruits and Vegetables. The list of members Dessert - Stick, Icecream, Donut, Cookie, Cake, Chocolate and Candy. Fruits - Peach, Banana, Grape, Apple, Lemon, Melon, Pear, Pineapple, Strawberry, Tangerine and Watermelon. Vegetables - Tomato and Aubergine. Were applied 30 times.

5. **Body Restriction** - The list of members Hand, Finger and Heart. Were applied 41 times.

6. **Topic Restriction** - The list of members Romance, Comic, Sport, Music, Drink and Education. Were applied 45 times.

7. **Symbol Restriction** - The list of members Geometric, Zodiac, Time, Word, Sign and Keycap. Were applied 155 times.

8. **Face Restriction** - The list of members Eye, Ear, Nose, Tongue, Boar, Mouth and Lips. Were applied 27 times.

9. **Face Expression Restriction** - The list of members Tear, Joy, and Smile. Were applied 14 times.

10. **Gender Restriction** - The list of members Man and Woman.

11. **Age Restriction** - The list of members Baby, Boy and Old.

Some of the emojis were potential candidates for an instance/individual without additional checks. An example of this is emojis representing a country flag. The logic applied here was that on all emojis with annotations *other* and $flag$ an object property $isAFlagOf$ was asserted with a country name as a value, from the list of individual equivalent to the class $Countries$. A similar approach was applied to all the animals who are insects, the $isInsect$ object property was set with a value from the list equivalent to the class $Insects$.

2.2.2 Properties

OWL refines the basic property type from RDF into two sub-types: object properties and data type properties. The difference between them is that an object property can have only individuals in its range, while a data type property has concrete data literals (only) in its range.[1]

2.2.3 Object Properties

Object property isA with a domain and range the main class $Emoji$ was created. The purpose of this property was to provide a clear description to which superclass each individual emoji belongs. For support of symbol restrictions object property $hasShape$ and $hasZodiac$ were created. To all plants emojis object property was applied if relevant. $isFlower$ and $isTree$ were properties to classes $Flower$ and $Tree$ respectively, with the

equivalent list of individuals maps to them. *Object* class has a property *isObject* with list of individual applied to all emojis when relevant.

The sample code below shows how the $Tree$ class and the $isTree$ object property are created using Jena API, assuming that *model* is a model into which the Emoji ontology has been read:

```
EnumeratedClass tree = model.createEnumeratedClass(uriBase +
EmojiProperties.PLANT_TREE, null);
tree.setLabel(EmojiProperties.PLANT_TREE, null);
tree.setComment("All trees emojis", null);
tree.setSuperClass(plant);
tree.addOneOf(model.createIndividual(uriBase
+ EmojiProperties.T_PALM, null));
tree.addOneOf(model.createIndividual(uriBase
+ EmojiProperties.T_EVERGREEN, null));

ObjProperty op = ObjProperty.getInstance();
op.addObjectProperty("isTree", baseEmoji, tree, true);


.................................................

/**
* Create object property, either functional or not.
* @param objectName
* @param domain
* @param range
* @param functional - true or false
*/
public void addObjectProperty(String objectName, Resource domain,
Resource range, boolean functional) {

property = model.createObjectProperty(uriBase + objectName, functional);
        if (domain != null) {
                property.setDomain(domain);
        }

        property.setRange(range);
        add(objectName, property);
}
```

Figure 2.1: Create Enumerated Class.

2.2.4 Data Properties

A number of data type properties were created. *hasUnicode* was applied to all emojis. In Jena API, when a data type property is asserted to a class, an instance/individual was automatically created. This was a big concern and the decision as that way the output from the program produced for each class a individual as well, which wasn't ideal. The *canBeEaten*, *isGeometricSymbol*, *isIdeographicSymbol* and *isZodiacSymbol* were created as a boolean property. The idea behind this was to clearly assert that the individual belongs to a certain category. The *showsTime* was created with a range String, it was used to assert a value to all individuals representing the a clock object with a particular time.

2.3 Manual Organization

A class in OWL is a classification of individuals into groups which share common characteristics. If an individual is a member of a class, it tells a machine reader that it falls under the semantic classification given by the OWL class. Classes may be organised into a superclass-subclass hierarchy, which is also known as a taxonomy. Subclasses specialise ('are subsumed by') their super classes. What we've done is define our semantic terms, or classes, in a hierarchy. In the semantic web world, this hierarchy of terms is called a taxonomy.[2]

Here's a graphical illustration of the taxonomy hierarchy defined:

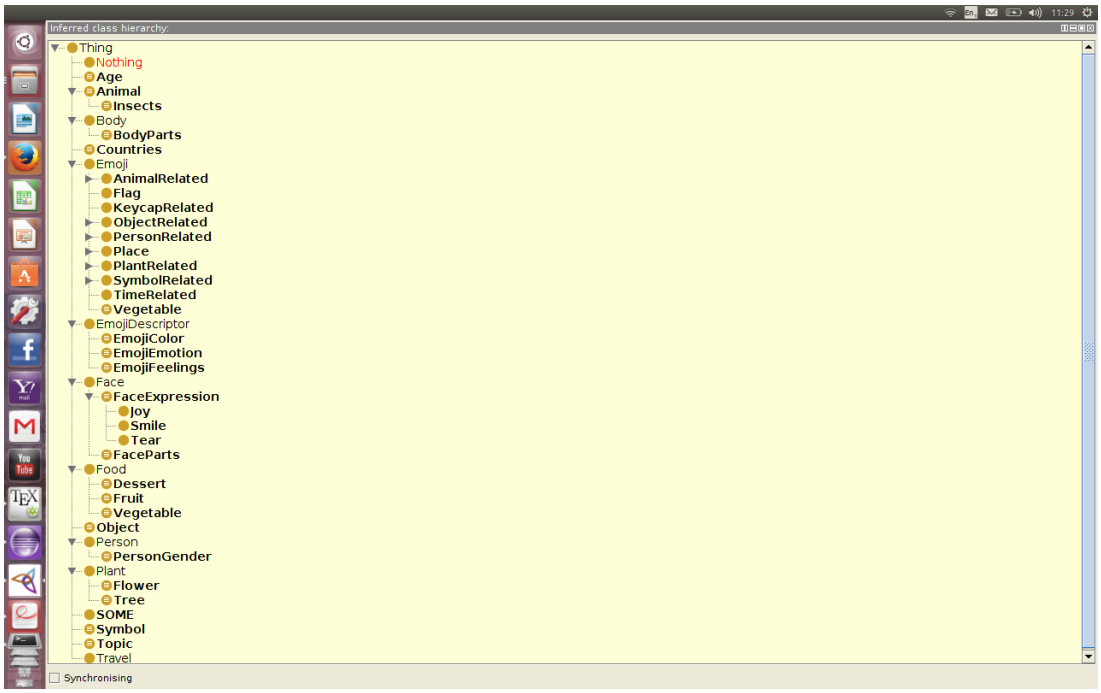

Figure 2.2: Emojis Taxonomy hierarchy.

The output owl file from the automation process gives us a good starting point to begin the manual organization of the Emoji ontology. Organizing the subclasses of *Emoji* was the toughest part of the project. For each class, comments were added with all available annotations. This helps us to apply the appropriate properties and restrictions to the class which were skipped during the program execution for one or another reason. New properties were created.

Number of things were amended in the Manual organization process. The first was to add all relevant disjoin class. Example is *BodyParts* was made disjoint with *FaceParts*, *Person* with *Animal*, *Desserts* with *Vegetable* and so on. Next was to go through the related subclasses of Emoji class and all classes that were well/fully defined were removed as a class and the appropriate object and data properties were added to the individual/instance. Example of it is under *AnimalRelated* we had *Camel* class with two different subclasses after the automation process. The difference here was just the name of the Camels, so new data type property was created *hasName* with range Name, these two camel classes were made individual of type *Camel* with data property *hasName*

and object property $isAnimal$ with value $Camel$. The same approach of removing the class and asserting the relevant properties was applied when appropriate. All well defined Cat Faces related classes were made an individual of type $CatFaces$, newly created with the appropriate data and object properties. The $Fish$ class was created and $BlowFish$ and $TropicalFish$ were made individuals of type $Fish$ with $hasName$ data property asserted on them.

Chapter 3

Conclusions

As a reference, the following search for emojis was used, http://emojinewssearch.com/. This was an inspiration to build an emoji ontology that could be used for a better search reference. The ontology built can be used to build a search engine not using the annotation or the emoji name, but using the object and data property values instead. If a search term is passed in the search engine it will be matched against all possible values of the object and data type properties. As an example, the emoji with class name $FaceWithOpenMouthAndColdSweat$ has the annotations (rushed, sweat, mouth, blue, open, face, person). In the ontology built, the restriction applied to it will contain a number of object properties with relevant values - isA - Person; $isFace$ - Face; $hasFacePart$ - Mouth; $hasColor$ - Blue; $hasFeelings$ - Cold. So this emoji will be return from the search if one of these property values is matched. To narrow down the search, multiple words can be accepted in the search engine, so less relevant results will be returned. This search engine will use all of the object properties from the emoji ontology, so UI representation will have multiple input fields. The name of these input fields will be equivalent to the object property and will accept the list of values for that property only. The search approach can be applied for a text substitution with emoji or just adding an emoji for visualisation purposes of the text content. The logic here is that the different words in a single sentence can be used to suggest an emoji based on the most matches. A possible application of this would be to pick emojis for a tweet created by a twitter

bot.

The heuristic used for solving the task of building an emoji ontology was as follows: The spreadsheet with all emojis data was very carefully studied, based on observation a decision was made to use solely the annotations for building the ontology hierarchy. For defining the individual emoji from the file, a number of object and data type properties were created. These were used to create an Anonnymous restriction for each entry from the spreadsheet.

In overall I want to state that I am very proud of what work I have done on the project and how much I learned from it. I can say for sure that the experience I gained working on this project will help me throughout my entire career as a software engineer.

Bibliography

[1] Jena API, *Jena Ontology API*,https://jena.apache.org/

[2] Semantic Web Primer, *Tutorial 4: Introducing RDFS & OWL*,http://www.linkeddatatools.com/introducing-rdfs-owl

[3] Pargfrieder,Karin., *Interorganizational Workflow Management: Concepts, Requirements and Approaches* Diplomarbeit, 2002, Diploma Thesis, Archive number-299489, http://www.diplom.de/.