Radboud University

Master Thesis Computer Science

Personalized life reasoning

Author:
Boy Boshoven
bboshoven@gmail.com

Supervisor:
Prof.dr.ir. Th.P. van der Weide
tvdw@cs.ru.nl
Second reader:
dr. P. van Bommel
pvb@cs.ru.nl

April 4, 2014

Abstract

People have more interaction with electronic devices each day, the data-footprint this leaves behind can be used to our advantage. By using the idea of a concept we can relate this data to a real-world concept. These real-world concepts are then described using the Resource Description Framework (RDF), by creating a set of triples (a directed graph) that describes the real-world concept. Depending on which source of data is used, different ways of defining the concepts are used. For textual data and sensory data the RDF-graphs representing the concepts look very different, as for one words describe the concept but for the other sensory data.

To be able to take advantage of this data-footprint left behind we can now use the concepts, which is called reasoning with concepts. The reasoning is done by defining a set of rules, where in these rules you describe what conditions individuals of related concepts should meet, when these meet, an action is performed. Actions are used to manipulate individuals or obtain information from individuals.

Using these rules we can then help automate certain tasks within a person's daily life, thus automatically using this data-footprint to our advantage.

Contents

1 Introduction

On an average day a person often has a lot of information needs for their daily activities. To fulfill these needs interaction with computer software on any electronic device is often required. This software can be at a lot of places, think of search engines online, navigation systems in cars, monitoring software on machines in hospitals or applications on mobile phones. There is often interaction required with this software to find what you need, although a lot of this interaction can be automated by predicting what the person wants to do or wants to know in certain situations. A system that is set up to make these predictions will require a lot of information about that person's daily activities and personal preferences. A great way to achieve this is by logging most activities the person is performing.

Currently, creating predictions using software is done in a few ways, from simple to more complex solutions. This goes from auto-complete user input based on previous input or input by others and is often used in search engines to solutions like Google Now, which try to predict and show information, like traffic information before going to work. However, it would be nice to know what the user is looking for so auto-complete becomes obsolete. Or, whenever somebody does not work one day in a week, it would be good not to show traffic information on those days. In present, existing solutions, either personal preferences are barely considered or are considered too often, which causes them to become either too obtrusive or too unreliable.

In this research the notion of reasoning with concepts (where activities are part of the collection of concepts) is explained where you can have personal preferences and customization to improve the assisting and advising abilities. To do this, the main question for this research will be: How can reasoning with concepts automatically help in one's daily life?

In this research we will first look at the state of the art research that is already been done related to this subject in section 2. Then we will look at the different types of data available to find details about the concepts we want to have and how to use this data effectively (section 3). When we know the available data types we can see which concepts can be detected and create a method defining these concepts (section 4). When we know how to define concepts, we want to use this to our advantage by reasoning with these concepts to help somebody in his/her daily life (section 5). After we know how to reason with concepts an example application is given in section 6 and when we have seen how this work in a practical application we can conclude this research (section 7).

2 Literature

The detection of concepts is something that has already been done in a lot of areas of research. However, the most prominent of the areas are: classification of user-data and search engines using semantic search.

User-data is basically a form of data related to the user interacting with an electronic device. For example:

- activity recognition using data collected by multiple sensors,
- pattern recognition using location data,
- profiling using tracking data.

2.1 Activity recognition

About 10 years ago activity recognition using certain hardware on one's body became a popular subject to study. Blum et al. [1] have shown that tracking human activities by using cameras, accelerometers, WiFi, GPS and audio to detect what the person is doing.

Around that time mobile phones also started picking up more hardware, of which the WiFi and GPS were one of the first additions. Embedding this hardware in mobile phones gives the advantage of not having to worry about any sensory devices placed on the body, as a phone is something a person carries around most of the time already. Liao [2] presents a way of detecting activities using this hardware by giving importance to location data by using machine learning and probabilistic reasoning. This results in the detection of a set of location-based activities that can be detected with a high accuracy. Location-based activities are activities a person is performing depending on where that person is, for example, a person is shopping because he is in a supermarket or a person is working because he is at his work.

After this, accelerometers were introduced in mobile phones (especially in smartphones) to detect things like screen rotation, but these sensors were also precise enough to be used as an accelerometer attached to your body like earlier studies [1] used them. Zhang et al. [3] use the accelerometer of the smartphone to recognize six activities using Support Vector Machine (SVM) classifiers, walking, posture transition, gentle motion, standing, sitting and lying. These are all very distinguishable activities when it comes down to recognizing them using accelerometer data. Bedogni et al. [4], however, limit their set of detectable activities to transport by car, by train or on foot. The focus there will be on transport by car and by train, because those are very similar activities when looking at accelerometer data.

The next step in the process of detecting activities has been the usage of data from multiple sensors to recognize the activities. Chon et al. [5] does this by using the accelerometer and gyroscope within the smartphone to track indoor locations (where there is no GPS available). Others [6, 7, 4, 8, 9] choose to combine multiple sensors to detect more specific activities, like jogging, driving or watching TV.

2.1.1 Algorithms

There are multiple algorithms that can be used to detect activities, but in most research the detection of activities is done using learning algorithms. Using learning algorithms for user-data is generally a good idea because it adapts to the data the user generates. However, learning algorithms usually focus on one type of data from one sensor, when you use the data from multiple sensors it is generally not possible to chose one algorithm that can analyze all the kinds of data, as GPS data looks very different from accelerometer data. In most research [8, 5, 4, 7, 6, 10, 11, 3, 9, 12] which make use of multiple sensors there is usually one algorithm assigned to one kind of data, and then there is another algorithm which then analyses the results of the different algorithms into one result. Or the other way around, one algorithm that looks at what type of data it is, and then selects the right algorithm for that type accordingly.

Hidden Markov Models are mostly used for pattern recognition, audio fragments [6, 1] or movement patterns (patterns in a series of coordinates) [2, 9] are mostly analyzed by them. Using audio fragments a context can be derived or excluded to enhance the precision in which activities are detected correctly. Movement patterns are used for a different purpose, namely predicting where the smartphone user is going next, indirectly this is of course gaining contextual information too. For accelerometer data however, SVM and Naive Bayes classifiers are often used, which are algorithms that can detect very specific activities and are able to differentiate between activities that look like each other (driving in a bus or driving in a car for example), given enough time to train the classifiers.

2.2 Practical applications

Since the first smartphones have been released with sensory hardware, especially the ability to get the location of the smartphone, there have been applications which attempt to do something with that information. All of these apps have their advantages and disadvantages, as all they all focus on

a certain feature at which they excel at. However, none of these really focus on activity/concept detection, they all focus on directly relating user-data to pre-defined actions to perform, like putting a mobile phone on silence at a certain location, or showing weather information for the area you are in.

Llama Llama [13] is an application which focused on location based data when it first was released. So, whenever the smartphone is at a certain preset location, a preset action is performed.

When using Llama you can basically set different areas you want the application to remember. You can ask it to learn a location when you are at the location itself. It will then save cell tower information for that location, whenever this is not accurate enough you can use GPS or WiFi locations at the cost of battery life. After these events can be defined where you can use the location information to switch between profiles when an event occurs. For example, when you leave the "Home" location you can change the profile to a profile named "Normal". For each profile certain phone settings can be changed, like volume control or the ringtone itself.

Tasker Tasker [14] is a tool that checks a fixed number of events and creates tasks that the phone should perform whenever a set of rules (rule-based approach) tells that a set of actions conform to the task should be executed. The set of actions that Tasker can perform is one of the biggest on a smartphone these days.

When using Tasker you mainly have the notion of tasks, and for each task you can perform a set of simple sub-tasks. These sub-tasks can be compared to a simple programming language that the developer of Tasker has created. An example of steps in which an alarm is played whenever you receive a text message containing the word "test":

1. Set the variable $\%BODY$ to what is in the predefined Tasker variable of what was in the last text message ($\%SMSRB$).
2. Perform a search on the variable $\%BODY$ and search for the word "test" and store these in the variable $\%MATCHES$.
3. Stop the script if no matches are found.
4. Play an alarm if a match is found.

Google Now Google Now distinguishes itself from other applications as it does not perform any action based on certain data, but it shows information based on data collected on Google services that can link to a Google account (like email, search and Android, which is an example of profiling using tracking data). So whenever the smartphone is at a certain location, useful information for that location is shown. Besides location based information it also uses accelerometer data to distinguish between walking, cycling or taking another method of transportation. It also checks emails and search queries performed by the smartphone user to give useful information on the smartphone (mail package tracking using a tracking number from an email for example). All these pieces of information shown are called cards, because they show up in a card format.

CommonSense CommonSense [15] offers a solution to track and analyze data collected by a smartphone using all available sensors. The data collected using the Sense application is stored on the cloud and is accessible or analyzable on a website made by the CommonSense developers. From every sensor or combination of sensory data new states for that data can be created, this will give the possibility to create virtual sensors. Activity tracking can also be done, this is basically the same as creating a state, but now with the type activity.

2.3 Resource Description Framework

Guha et al. [16] use a technology called Resource Description Framework (RDF) [17] to model information found on the web in a structured manner. RDF does this in the form of triples, these are expressions build out of three parts, a subject, a predicate and an object, in which the subject is related to an object using the predicate:

$$subject \xrightarrow{predicate} object$$

Using triples as the basic components, a model of almost any information can be made by combining multiple triples for which each triple describes a piece of information. RDF defines defaults using RDF Schema (see table 1), which (similar to what XML and XML Schema do) describes what is possible in specific RDF models. Like in XML, RDF uses namespaces to differentiate between elements from different vocabularies of definitions, making them unique. For RDF this namespace is http://www.w3.org/1999/02/22-rdf-syntax-ns# and for RDF Schema it is http://www.w3.org/2000/01/rdf-

schema#. To denote an element e in either of these namespaces it is conventional to write: rdf:e for RDF or rdfs:e for RDF Schema. However, for readability reasons no namespaces will be used throughout this research.

Element	Description
Resource	Everything defined in RDF are member of the class *Resource*
Class	Is the type/category of a newly defined concept
Property	A property is the type of resource that denote a property
Literal	Literals are resources that denote a fixed value (for example strings, numbers, dates)
$x \xrightarrow{type} y$	x is of type y
$x \xrightarrow{domain} y$	x belongs to domain y
$x \xrightarrow{range} y$	x has an element of concept y
$x \xrightarrow{subClassOf} y$	x is a more specific concept than y
$x \xrightarrow{subPropertyOf} y$	x is a more specific property than y
$x \xrightarrow{label} y$	x denotes a human-readable name of y
$x \xrightarrow{subClassOf} y$	x is a more specific concept than y

Table 1: Available elements in RDF

There are many syntaxes available for RDF of which each focuses on different elements:

- Turtle; compact and easy to read
- N-Triples; compact and easy to parse
- JSON-LD; JSON-formatted triples
- RDF/XML; XML-formatted triples

An example in Turtle would look like this:

```
@prefix eric:    <http://www.w3.org/People/EM/contact#> .
@prefix contact: <http://www.w3.org/2000/10/swap/pim/contact#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

eric:me contact:fullName "Eric Miller" .
```

```
eric:me contact:mailbox <mailto:em@w3.org> .
eric:me contact:personalTitle "Dr." .
eric:me rdf:type contact:Person .
```

where you can see that it is about the person named Eric Miller, along with his email and personal title. The name, email and personal title all are properties of the instance called "me", because "me" is defined for the namespace called "contact", in which the properties "fullName", "mailbox" and "personalTitle" are available. To find the properties that became available there is an RDF file that describes these properties for "contact", located conveniently at the URL defined as the namespace prefixed "contact": http://www.w3.org/2000/10/swap/pim/contact#.

2.3.1 Web Ontology Language

In RDF almost anything can be expressed due to the flexibility of the language, this also means that besides RDF Schema there can be more extensions, as long as it is defined in its own namespace. Web Ontology Language (OWL) [18] does this and creates the ability to reason with the situations defined in the RDF-models, making OWL a Description logic language. Because OWL is just an extension on RDF, it is just an addition of elements to RDF (see appendix A) which are used for reasoning with the concepts you have already defined in RDF.

2.3.2 SPARQL

SPARQL [19] is a query language created to be used on situations defined using RDF. SPARQL can do all the things expected from a query language:

- SELECT; extracts values from an RDF graph
- CONSTRUCT; creates new RDF from a set of results
- ASK; provides a boolean result for a query
- DESCRIBE; extracts an RDF graph from a query
- DELETE; deletes values
- INSERT; inserts values

An example of a query used on the example defined in section 2.3 which will give a result in table format (see table 2) looks like this:

```
PREFIX contact: <http://www.w3.org/2000/10/swap/pim/contact#>
SELECT ?fullName ?mailbox
WHERE {
  ?x a contact:Person .
  ?x contact:fullName ?fullName .
  ?x contact:mailbox ?mailbox .
}
```

fullName	mailbox
"Eric Miller"	⟨mailto:em@w3.org⟩

Table 2: Results of the query

2.4 Discussion

Each of the subjects in the previous sections will have its use in this research and each of these subjects will have their use and limitations.

2.4.1 Activity recognition

Activities in itself can always be seen as concepts, this makes the field of activity recognition very useful for this research, since we want to reason concepts. Using these activities creates a lot of possibilities, because they are concepts that are closely related to the user performing these activities, thus automatically creating personalized concepts.

2.4.2 Applications

The applications all have their own advantages of which some are usable for this research. Generally, applications like Tasker only actually do something whenever its user creates actual tasks. This method of personalizing, by actually letting the user tell what the application should do, can become very powerful when used in conjunction with concepts.

Other applications, like Google Now, are very powerful in information representation, using its cards system it has a great method of informing the user what is going on.

2.4.3 Resource Description Framework

This method of storing and representing concepts is very flexible, as it is basically XML (or variants of XML created for RDF, like the languages Turtle, JSON or N-Triples), and expandable the way you want to expand it,

as the definitions the terms that can be used in RDF are defined using RDF Schema (variant to XML Schema). The possibility of having an ontology (your own namespace) helps a lot as well, because it creates the possibility of creating your own domain for your own concepts, in which you can create concepts of the same name as other ontologies.

3 How and why is data obtained?

There are a lot of kinds (types) of data which can be used in an automated informative or helpful way for a person. To collect this data often sensors or logs have to be collected and if needed, converted. Sometimes the data is already available in its correct format for an algorithm to use. Because in this research we want to be informative or helpful to a person, the types of data are kept limited for that focus, but can be expanded for alternative usage.

3.1 Types of data

Most of the data about a person, can be gained from that person. This makes the sensors in a smartphone a great choice to obtain data from. Another big source is a person's digital activity, not only on a smartphone but also a computer at home or work.

3.1.1 Location data

Location data can be gained in multiple ways, most important of these are: GPS, cellular network, WiFi, IP location tables, HTML5 location sharing.

Format: $(longitude, latitude)$ a pair that gives the longitude and latitude coordinates.

GPS The GPS gives a very accurate location (within 5 meters) of the GPS device. GPS can only use satellite locations whenever nothing is blocking the signal, in or in between buildings GPS can be unreliable.

Cell towers The cellular network (usually used by phones) can provide a device location, this is derived from the positions of cellular towers, whenever there are three or more towers nearby the location can be calculated accurately (within 50 meters) using triangulation. However, when there are fewer towers in range of the device, the accuracy decreases rapidly.

WiFi Through WiFi a device location can be found as well. Whenever the location of the router to which your are connected is known, you know the proximity of that device. Multiple providers, like Google provide a database of routers and their locations, so whenever a WiFi node from one

of these databases is nearby the phone, you know the location of the phone as accurate as the range on the WiFi signal.

IP location tables There are databases where the location of certain IPs are located. Whenever an IP and location are fixed, this can give a very accurate position.

HTML5 location When a browser supports this HTML5 feature the browser offers a feature to request the location of the device the browser is on. It depends on the browser on how the location is acquired.

3.1.2 Body movement data

Body movement can only be gained using accelerometers, which can be placed on different parts of the body, but there is also one in most smartphones. There are single-axis accelerometer, which only detect movement in one direction, but more common are the multi-axis accelerometers, which detect movement in multiple directions, usually three.

Format: (x, y, z) for a 3-axis accelerometer, where x, y, z give the acceleration in their respective directions (in m/s^2).

The direction can be gained using a gyroscope, which can give the rotation of the gyroscope on multiple axis, often three axis are used like an accelerometer.

Format: (x, y, z) for a 3-axis gyroscope, where x, y, z give the acceleration in their respective directions (in rad/s, radian per second).

3.1.3 Sound

Sound can be gained from using microphones, almost every smartphone has a microphone which can be probed to record sound.

Format: x in dB.

3.1.4 Temperature

The temperate can be gained from a thermometer, these are included in the newer smartphones. However, thermostats and other devices can also acquire the current temperature.

Format: c in degree Celsius (°C).

3.1.5 Light

Measuring light levels is usually done using light meters, however, using any camera can also be used to detect levels of light in a less precise manner, by analyzing images or videos it records.

Format: l in lux.

3.1.6 Nearby devices

Detecting nearby devices can be done multiple ways, whenever a smartphone is used, Bluetooth devices, WiFi nodes and NFC chips are popular ways of detecting nearby devices, as those all have short range on their wireless communication.

Format: all have identifiers to know which device is who, WiFi nodes can use names, but is only uniquely identified with a MAC address. Bluetooth uses an device ID. NFC chips do not necessarily have identifiers, however, software solutions often use identifiers, which then could vary depending on what the NFC chips are used for.

3.1.7 Usage statistics

Using information about what application a person is using on a computer or smartphone, one can often derive what the person is doing.

3.1.8 Textual

Textual data is another kind of data as this does not come from any sensor, (useful) textual data is often produced by humans in the structure of words and sentences, saved as a document. Using this structure we can derive what that document is about and what the important elements of that document are using multiple techniques. From this conclusion we can use these documents to find information about the elements that they contain.

Format: HTML (webpages), PDF, Word, Plain text, and more of these standard formats to save a text document in a structured manner.

3.2 Combining data types

Some data types on their own are often less useful as when they are combined with another. The most obvious type which can be used with almost any other type is time. Keeping track of when a type had certain values is the most important combination to have, this allows most classification algorithms to work and allows us to combine certain values into a new data type. This is of course only true if you can access the values a certain type had previously, so the storage of all values is important (also known as data logging).

3.2.1 Virtual data types

The notion of virtual data types is combining two values of any type of data (either the same or a different type) and combining it into one new value, creating a new data type.

Historical data Whenever you are able to combine time with the values that you have recorded you can often combine data values previously recorded with data values you are recording at the moment. This gives the possibility to create a new data type out of two values of one data type. An example of this is when you keep track of location data, you have the option to calculate the distance between the two points of location data and you can calculate the difference in time between those two points. Whenever you have the distance and time traveled, you can also calculate the speed you were going during that period of time and is an indication of your current speed if you take the the current and previous positions. This example creates a new data type which can be called speed.

Combining data Another way of combining data into a virtual data type is taking multiple values from one sensor, the accelerometer for example, and combine those values into a new single value. Taking the accelerometer as an example calculating what force is working on an accelerometer by calculating the Euclidean norm for the vector of the acceleration on the multiple axis the accelerometer returns. This will give a generalized form of the direction and acceleration working on the accelerometer, if the device with the accelerometer would lay still somewhere on planet earth, the only force affecting the device would be the gravity (9.8 m/s downward).

Not only can you combine data from the same sensors, but also from multiple sensors. For example in smartphones, they have no compass, but

can still get the direction of which the smartphone is laying/moving by combining the data of the accelerometer and the data of the gyroscope, this can then be transformed into a general direction of which the smartphone is pointing.

3.3 Data logging

Being able to get the most out of the available data for a person, it is essential to log all or most of it. Whenever you have historical data available it creates the possibility of detecting patterns in your data which is essential to be able to recognize concepts within the data using machine learning algorithms. Also, it gives the possibility of creating a virtual data type out of data values of the same data type, like the usage of location data to calculate speed.

Having all possible values for data at any given time gives the possibility of having a data set and data points, resulting in the terminology used like in table 3.

Term	Description
Data point	A value of some type of data, at a given time
Data set, data sample	A set of data points, from one time to and including another time ($[t_s, t_f]$, where t_b is the start time and t_f is the finish time).
Data	A general notion of data, referring to any or multiple data point(s)
Data type	A type of data, like described in section 3.1

Table 3: Terms related to data used throughout the research

3.4 Data patterns

Whenever you have the ability to extract previous recorded values for certain data types (by logging them), you have the ability to find patterns in them. However, patterns in data is not only limited to loggable data, basically any data that can be analyzed could also contain patterns, examples are webpages, books, videos and images.

Patterns in data can be useful because they tell something about the data which helps the process of identifying what that data is about. Examples are:

Body movement data (accelerometer) For movement data, patterns can be the length of the period of the oscillation in a set of data. This might tell something about the movement of the person, as if the period is shorter, it is likely the person is doing a repetitive movement at a fast pace.

Then there is the average maximum and minimum value within a data set, which tells something about how "big" the movements are. As when I am walking there are little forces detected by the accelerometer, but when I start running these forces become bigger (a bigger acceleration in a certain direction).

Textual For textual data a pattern can be the number of times a word occurs in a document, but not in any other, then it is more likely that the document is about that word. The tf-idf algorithm works using this principle and can be a useful pattern to have for textual data.

For machine learning algorithms these patterns are often called features, which these algorithms use to compare two data samples with each other.

4 Conversion from data to concept

To make the collected data useful it is good to convert them to the notion of a concept, these concepts are very similar to the real-world concepts used by Guha et al. [16].

4.1 What is a concept?

There are many methods of describing concepts in a formal way, depending on the field of research. For object oriented programming languages, a single class represents a concept. For data storage solutions (like SQL databases) tables represent concepts. For information retrieval, specifically semantic search, RDF was introduced to describe concepts. RDF was then extended by for example RDF Schema and OWL to add additional functionality.

Using RDF, there is certain a terminology which is used to describe aspects of a concept, an overview of these terms have been given in table 4, all of these terms can be described using triples.

Terms	Description
ontology	collection of concepts/classes
concept, class, type	a description of a real-world concept, a collection of individuals
individual, instance	one identifiable realization of a concept
value, literal	similar to an individual, except there is no emphasis on the concept
relations	describes how classes and/or individuals are related to one another
predicate	the name of a relation, telling something about it
attribute, property, role	a relation between two individuals or an individual and a value, often describing aspects of an individual

Table 4: Terminology used in ontology languages

Triples are three elements that represent information, often a sentence (see section 2.3). A triple consists of a subject and object, linked by a predicate:

$$subject \xrightarrow{predicate} object$$

For example, when looking at the sentence "the sky is blue" could be translated to a triple where you have "the sky" as the subject, "blue" the

object and "is" the predicate. The triple is about the sky, where the predicate tells something about how the object relates to the subject, in this case the sky is blue.

Using this we can translate any real-world situation to a list of triples, which are a representation of that situation. These triples can then be combined in a directed graph. This gives us:

$$G = \langle N, E \rangle \tag{1}$$

where graph G is the combination of a set of nodes (N) and a set of directed edges with a predicate (E).

Edges An edge is a relation between two nodes, with a direction and a predicate. Most predicates related to the edges are dependent on the mapping of the real-world situation, however, there are a few predefined edges in RDF (see section 2.3), for which the most important for this research are:

Edge	Description
$x \xrightarrow{type} y$	x is of type y
$x \xrightarrow{domain} y$	x belongs to domain y
$x \xrightarrow{range} y$	x has an element of concept y

Nodes The nodes in the graph can represent a lot of things, depending on what edges are connected to it. For example, when looking at this situation:

$$x \xrightarrow{type} y \xrightarrow{type} z$$

when you only look at the first triple ($x \xrightarrow{type} y$), x will have the type y. But when you look at the second triple ($y \xrightarrow{type} z$), y gets a different meaning, namely: type z. Combine this and y is the type of x and has the type z, giving it multiple purposes. By having multiple edges going to or coming from a node, it describes the node more precisely, which leads to a more detailed description of the real-world concept the node is intended to describe.

For this research, there are also a few important nodes which are already defined in RDF:

Node	Description
Class	Is the type of a newly defined concept
Property	A property is the type of a node that denote a property
Literal	Literals denote a fixed value (for example strings, numbers, dates)

4.1.1 Concept modeling

Using a graph to model a real-world concept two things are required:

- Intension; the definition of a concept, defining what properties and relations are allowed
- Extension; the description of our real-world instance/individual of the concept according to the intension

Properties Properties are needed to create defining elements of a concept. Defining a property can be done by creating the following triples:

1. The subject is a property, some p. So it has the superficial type "Property".
2. p has a domain in which it can be used. This is usually the concept we want to create our property for. (some c)
3. p has a range to what kind of literal it relates to (for example string, number or date).

These can be translated into the following triples:

$$p \xrightarrow{type} Property$$
$$p \xrightarrow{domain} c$$
$$p \xrightarrow{range} Literal$$

An example of a concept with a property could be the concept dog-breed with an individual of that concept, say the breed Newfoundland. If I want to say the actual name of the breed Newfoundland is "Newfoundland", I could create the following set of triples:

$$Dog \xrightarrow{type} Class$$
$$name \xrightarrow{type} Property$$
$$name \xrightarrow{domain} Dog$$
$$name \xrightarrow{range} string$$
$$Newfoundland \xrightarrow{type} Dog$$
$$Newfoundland \xrightarrow{name} "Newfoundland"$$

These triples can be displayed like we want to, using a graph where the predicate in the triples define the edges and the subject/object define the nodes (see figure 1).

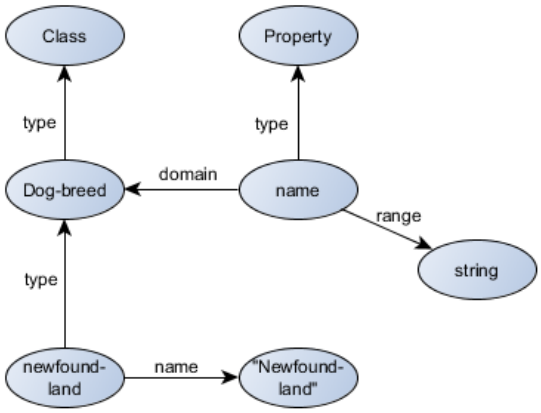

Figure 1: Dog-breed example

4.2 How do concepts relate to each other?

Being able to describe any concept in the way you want to, it gives a lot of possibilities. However, in a lot of cases you want to know what the relation of a concept is to another concept. Describing a relation for a concept is done the same way properties are defined for concepts. The only difference is that the property is now related to a range that describes a concept:

$$p \xrightarrow{type} Property$$
$$p \xrightarrow{domain} c$$
$$p \xrightarrow{range} d$$

here, a property p defined for concept c, which is related to an individual of concept d. This property is a one on one relation between an individual of concept c and an individual of concept d.

Whenever you take c and d as the same concept, and you define a relation between c and d then it will be a recursive relation where individuals of the same concept are related to each other. With recursion, more complicated situations can be modeled, an example of this is given in figure 2.

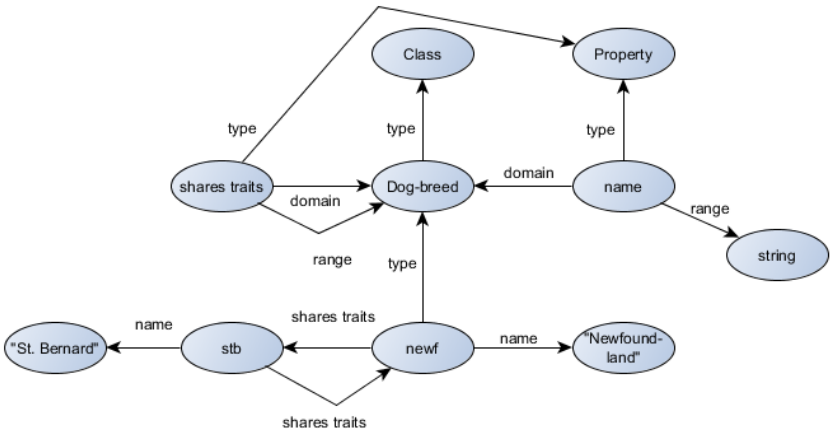

Figure 2: Example that uses recursion

4.3 Finding concepts

The ability to find concepts from data depends on the type of data you use to recognize this concept from. RDF has been been created to be able to do this for web pages, however this is not the case in more different types of data, like images, video or movement data. A lot of other methods have been created to recognize concepts out of these types of data, for example machine learning algorithms are often used to derive activities from body movement data, in which the activities can be seen as concepts.

Supervised Most of the concept recognition algorithms in this research are supervised algorithms. Supervised algorithms attempt to find a concept and its relations within a given set of data. So, whenever we know how the concept (and its relations) is structured within a set of data, a supervised algorithm will try and find the same structure in another set of data.

Unsupervised Unsupervised algorithms do the opposite as supervised algorithms as they try to find a structure in a set of data. Because unsupervised algorithms are able to do this they can help supervised algorithms by finding the structure the supervised algorithm needs to be able to recognize a concept. For example, an unsupervised algorithms are able to recognize certain elements on a webpage and tag it with metadata (web scrapers), which then can be used by a supervised algorithm to match with a concept. Unsupervised algorithms are also used to predict what future sets of data are going to look like, in this way they attempt to predict what is going to happen next. This is something that is used for weather forecasting.

4.3.1 Semantic search concepts

Concepts related to the field of semantic search are defined using the same methodology as shown in section 4.1. For semantic search storing these graphs is important so they become easily accessible and searchable. To be able to search through them serialization formats are used, of which RDF/XML, N-Triples or Turtle are popular examples.

Using these formats, query languages are created to look through this serialized content to find concepts, relations to concepts or properties of concepts, SPARQL is the language that is often used for this.

Data types used: Textual

4.3.2 Machine learning concepts

Machine learning algorithms are used to find similarities in two different sets of data of the same type. Body movement data is a great example for this, as if you are walking, the data measured with an accelerometer over a period of time will never look the exactly the same, but just very similar, this makes machine learning approaches very suitable to detect concepts from body movement data.

The general approach of machine learning algorithms is to compare data where it is known what the data represents, to unknown data set from which

you want to know if it compares with any known data set. This results in either no classification at all or a classification to a known concept. If the algorithm results in no classification but should have, then the data can be annotated to know what that data represents in the future. This way the algorithm "learns" more representations from this data the more it will be used.

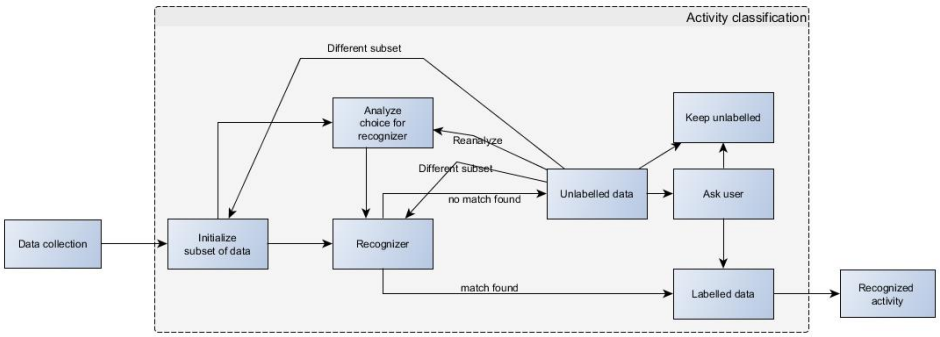

Figure 3: Concept recognition by machine learning algorithms

The most popular approaches to machine learning for recognizing concepts are Support Vector Machines (SVMs), Naive Bayes and variants on these approaches. These approaches make use of features extracted from data sets, the usefulness of features is decided from what type of data they are extracted. For body movement data you have certain features which can be useful: total data sample time, average acceleration on each axis, maximum/minimum acceleration on each axis and more. But when you look at usage of these algorithms in textual classification, features like the average acceleration do no apply. In textual classification the words the document contains are the features, but it can also include the document length or the number of occurrences of certain words. Textual classification is often used in spam filtering. (as seen in section 3.4)

Finding concepts in certain types of data, like body movement data, is very different from finding them using textual data. When looking at RDF to describe the relation between movement data and a certain concept, it might be difficult to realize this for these types of data. This can be solved by saving the movement data of all the axis related to a certain body movement concept, by storing the sensory data and the time frame of that sensory data as a property of an individual. For example, when you have an individual called walking (with walking as the concept as well), there will be a relation

between that individual and the values of body movement data which have described the idea of walking in the past, similar to how SQL databases do this.

In figure 4 we can see an example of how a machine learning concept can be represented using triples. We have the machine learning concept c, which has an individual i. For individual i a property p which has a relation to value v which has the type "ML value property". An "ML value property" concept is a concept which always has a value of a literal type and a timestamp for when it was stored.

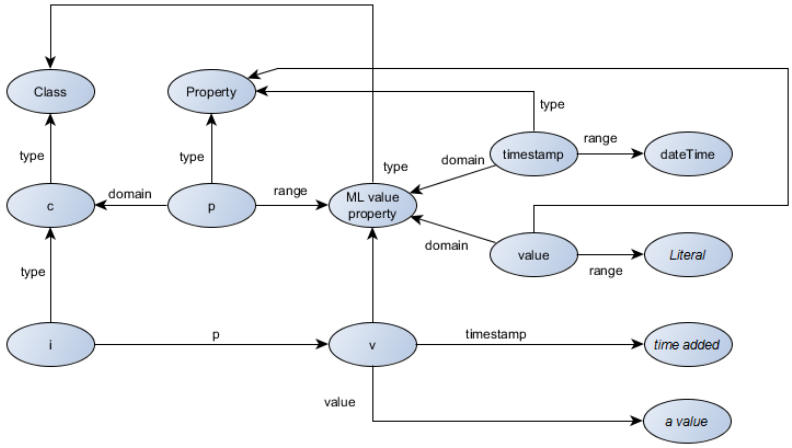

Figure 4: Machine learning concept

Data types used: Textual, body movement, location data, sound, temperature, light, nearby devices, usage statistics

4.3.3 API concepts

An API concept is a concept that is not defined locally with the algorithm, but actually represents the concept in question. Examples of this can be a concept bus, as an individual "line 300" or a concept heater, as an individual "my heater".

Bus While taking a look at busline 300, which could be the bus I want to take to work, then I might want to know something about that busline. For example, how busy it is at the moment or if it is delayed. Using an

API this can be requested at real time, as long as the API can provide the information (or property of individual "line 300"). As it currently stands for many countries, an API to do this to see if the buses are delayed is available in many countries. However, to see how busy it is in a bus, it is not (yet) available in most countries (if not all).

Heater The same situation as with the bus can be created for "my heater", which actually represents my own heater at home. When my heater has an API available, not only information can be provided, like with the bus. But controlling the heater through an API can be possible as well, where as changing the temperature is the most obvious thing you want to control.

API concepts are very different from other concepts, as they do not have actual property-values stored locally, but rather the URL or method to call to get or set a certain property for an individual of that concept. This makes API concepts the only means of controlling real-world concepts as they can change property values. In the definition of an API concept a method of accessing the real-world concept or property of a concept has to be stored, in figure 5 a method of doing this is shown. Where you define a concept p of type "API Property" which has a property that tells where the API is located. Now it is possible to use the concept p as a relation to the concept c, so any individual i of type c can have an edge with predicate p. Then, the value v will be the value obtained from the API through the API location defined on p.

Data types used: none, used to represent real-world concepts

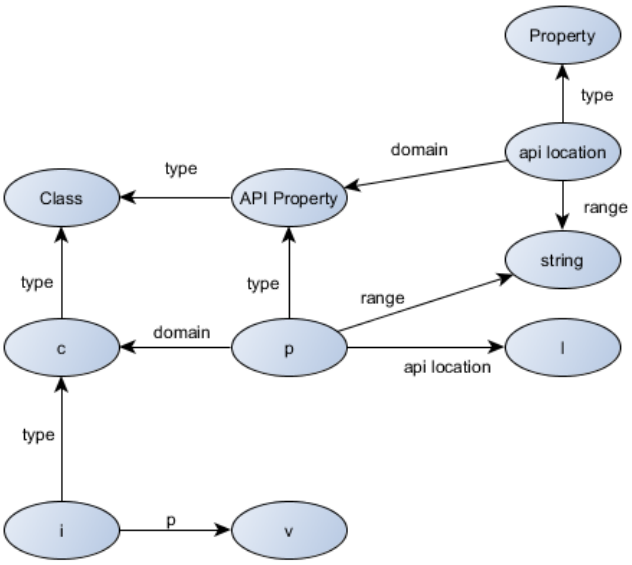

Figure 5: API concept

5 Reasoning with concepts

When you have defined a lot of concepts, we want to do something with these concepts. Semantic search looks for information within these concepts and its relations, but we can do more with these concepts, by reasoning with concepts. The process of reasoning with concepts will be explained in this section.

5.1 What is reasoning with concepts?

Reasoning with concepts is a method of talking about concepts in a logical way. OWL (see section 2.3.1) adds some of the functionality to RDF which helps with this process, because OWL can define restrictions, axioms and facts about the concepts we have defined. By doing this we can infer some non-trivial information automatically. Using OWL can merely help to reason with our concepts, often description logic languages like OWL are not needed.

Given the following statements, an example of this inference process is given:

1. Cat owners have cats as pets.
2. "has pet" is a sub-property of "like"
3. Cat owners must like a cat.

Here you infer from the sub-property that cat owners like their pets and conclude, whenever you want to know, "Do cat owners like their cats?", you can infer it from the first two statements (mentioned above).

Elements in OWL, like a sub-property, which is used as the predicate of a triple give a restriction to the subject, in this example the subject "has pet" is always a sub-property of "like", in which the restriction lies in that having a pet implies you like the pet.

5.2 Why is reasoning with concepts useful?

There are a lot of situations in which the detection of concepts related to the user who wants to reason with these concepts. In this research the notion of rules will be used to reason with concepts, a rule either fulfills an information need or is does information manipulation. For both of information representation and information manipulation methods have been developed already, which can be seen in the practical applications discussed in section 2.2. For example the following set of rules:

1. Do I do enough to keep my condition up?
2. I want to be at work at 0900.
3. Is my bus delayed?
4. Make sure the heater is turned down when nobody is home.
5. An email should be read within one hour of arrival.
6. I do not want to be disturbed when I am in a meeting.

All of these rules have a supporting impact on that person's life, but in the case of some situations it has also an impact on external factors. Take for example the fourth situation, whenever somebody forgets to turn down the heater it not only increases their bills but it is also indirectly bad for the environment.

Rules 1, 2, 3 and 5 are examples of rules that fulfill an information need, whereas rules 4 and 6 are examples of information manipulation. An information need gives information about concepts, for example in rule 3, where the user wants to know if his/her bus is delayed. Information manipulation changes values that are related to concepts, for example in rule 4, where the heater is turned down whenever nobody is home.

5.3 How to make reasoning with concepts feasible?

OWL, the description logic language we will use, is able to define the example rules mentioned in the previous section (section 5.2). However, there are a lot of possible ways to define these rules using OWL, this makes it difficult to create an universal method of letting a algorithm perform the things defined in these rules. Not limiting the amount of concepts and what is allowed with these concepts in a specific rule can lead to an uncomputable algorithm.

Being able to keep the algorithm computable some limitations are required. To do this, the syntax for a generic rule will be given, which limits the expressiveness for a rule, but keeps them computable.

The algorithm presented in this research consists of a few important parts:

- Rules - The translation of what the user of the algorithm wants and what the algorithm should do
- Automation - The actions the rules can perform
- Distribution - Whoever performs what within the algorithm

5.3.1 Rules

The rules provide an interface between the user and the algorithm, defining a rule has to be simplistic enough and still be able to let the algorithm work with the concepts that we want the algorithm to interact with.

Syntax To do this a predefined format for a rule is used, where there is a limitation on the number of individuals used in each rule, to keep the algorithm computable and to make it easier for the user to define a rule. The syntax for a rule looks as follows:

1. `when` *individual* i_1 *condition* c_1
2. [`then when` *individual* i_2 *condition* c_2]
3. `then perform` *action* a `using` (i_1, i_2, i_3, i_d, t)

where i_1 and i_2 are individuals of certain concepts, both i_1 and i_2 have a relation to c_1 and c_2 respectively where $c_{i \in \{1,2\}}$ is the condition that must hold for the algorithm to continue on that rule. The second line is between brackets because it is optional, when left empty the ε is used. a is the action attached to the rule, which can be any of these: a resulting value, a property change for individuals, a query related to the individuals. i_3 and i_d are the individuals where the action is performed on, where i_3 is optional and i_d is usually the individual where information for a query is displayed on. t is the action type.

To give an impression on how this looks on realistic rules two examples, one including the optional part and one excluding the optional part will look as follows. The rule "I do not want to be disturbed when I am in a meeting." could look like (could, because another person might define this in another way):

1. `when` *my calendar* *in a meeting*
2. ε
3. `then perform` *silent mode* `using` (*my calendar*, ε, ε, *my phone*, *state*)

The one including the optional part of the syntax, "I want to be at work at 0900.":

1. `when` *my location* *near bus-stop*
2. `then when` *busline 300* *delayed*
3. `then perform` *notification on how much the bus is delayed* `using` (*my location*, *busline 300*, ε, *my phone*, *at start*)

Semantics The resulting semantics for the steps of each part of the rules are:

1. Checks if condition c_1 hold for individual i_1.
2. Checks if condition c_2 hold for individual i_2, whenever i_2 and c_2 are defined.
3. Perform the action, involving the individuals i_1, i_2, i_3, i_d as the action-type t.

Conditions The conditions are always related to the individual in the same part of the rule. Within conditions we can define a comparison with a property-value or relation of the individual. Basic arithmetic is also allowed within these conditions so more advanced conditions become available.

Rule priority Whenever two rules use the same individuals, for which at least one of the rules changes a property of an individual to a different value, it might happen that the changes are conflicting with each other. To solve this problem a rule priority can be added, so that only the value defined in the rule with the highest priority applies, or that a property-value is only read when another rule did not change it (yet).

Rule definition Because the concepts we want to reason with are defined in RDF, we can also define our entire rules in RDF, so existing advantages of using RDF still exist when working with these rules. A translation from the syntax defined earlier to an RDF definition has been given in figure 6 on page 39 (intension) and figure 7 on page 40 (extension). For the intension the green nodes are of type Class and the orange nodes are of type Property, these nodes (and relating edges) are left out for so the graph is easier to read.

5.3.2 Automation

The automation in each rule is the action, these actions can be a resulting value, a property change for individuals or a query related to any of the individuals.

Resulting value Whenever an action is a resulting value, the rule can also be seen as a new individual of an unspecified concept. This allows for the possibility of recursion, thus using rules as individual for another rule.

Property change An action that changes a property allows for the controlling of real-wold concepts. For example, the heater that is turned off, or the garage door to be opened or closed. Because description logic does not give the ability to change and control an individual related to the action, a method of doing this has been introduced in section 4.3.3.

Query A query can fulfill an information need for the subject interacting with the algorithm. There are many query languages which are compatible with RDF, the language we use to describe the relations of different elements within our rules. However, there might be elements which are not queryable locally, and have to be queried through an API, which is not something these languages can do, so whenever a resulting query gives an API as a result, the monitor can then query the result from the API.

5.3.3 Distribution

There are multiple methods of evaluating the rules, each of these methods have their advantages and disadvantages.

Distributed In a distributed method, whenever an individual in a rule represents a real-world device that is able to evaluate the condition or action related to the rule, we can let that device do the evaluation and then report to other devices whenever the evaluation or action is done. This way each device only has to keep track of its own actions and each device will need very little computing power to perform the actions it is capable of. However, every individual will need to be a device and a protocol of communicating between the devices is required.

Advantages:

- Modular, if one individual fails, the rest can still continue
- Each individual only needs to know about its own actions

Disadvantages:

- Communication protocol needed
- Overhead as there is more communication needed and each devices has to run the algorithm, even though it is just a part of the algorithm.
- An individual needs to be able to run the algorithm

Centralized A centralized method, where we have one device that controls all individuals and performs all actions defined in the rules. This would require only one device that evaluates all the rules, this is also the only device that would need to know about communicating with other devices, so there is no need for a communication protocol.

Advantages:

- Easy to install, because only one device needs to run the algorithm
- No communication protocol for individuals needed

Disadvantages:

- If the centralized device fails, the whole algorithm will not run

In this research the centralized method of evaluating the rules will be taken, simply because the distributed method has the requirement that all individual need to be a device, which is not something we want. In this centralized method the device that controls all individuals will be called the monitor, because it has to monitor (and sometimes change) the behavior of individuals using the conditions and actions. When the rules only contain individuals that are (representing) devices the distributed method should be considered, because having a modular system is a great fail-safe when one device fails.

5.4 The reasoning algorithm

The input and algorithm are split up in the following steps:

1. Convert the information interaction of individuals to rules (*Rules*)
   (a) Create a description of an information interaction in a natural language
   (b) Mapping of sentences to elements in the rule
2. Have the monitor validate the first element of each rule on an interval (*eval*)
   (a) If the condition of the first part of the rule validates to 'true', perform the rest of the rule
      i. If the condition of the second part of the rule exists, and does not validate to 'true', continue to the next rule

ii. If the second part of the rule does not exist, perform the action

(b) If the first part of the rule does not validate to 'true', go to the next rule

```
function eval (Rules) {
  foreach (R in Rules as (i1, c1, i2, c2, a, i3, id, t)) {
    if (c1) {
        if (i2 ∧ ¬c2) continue;
        a(i1, i2, i3, id, t)
      }
  }
}
```

5.4.1 Rule execution

As seen in the algorithm the execution of a single rule, which is the inside of the foreach-loop, is done in two or three steps depending on if the second rule is defined. The steps that take up significant computing time are the evaluation of the c_1 condition, c_2 condition and the action a, where c_2 is not executed in some cases. The evaluation of these conditions consists of evaluating a predicate defined in a description logic language (which is translated from the definition the user sets in the rule). Predicates defined in the rules are simple queries to see if a certain individual has a property or relation with another individual. However, when an individual is a API concept a simple query will not suffice, as the individuals for these concepts will need to make an API request to get the required value.

Because we are working with a rule that is defined in RDF-format, we can use SPARQL (section 2.3.2) to query the different elements from the situation the rule describes, but also the parts of the rules. Say we want to find what the conditions and actions of all rules are, we can perform the following query:

```
PREFIX rule: <http://www.example.org/ExampleNamespace#>
SELECT ?whenCondition ?thenWhenCondition ?action
WHERE {
  ?r a rule:Rule .
  ?r rule:when ?whenCondition .
  ?r rule:thenWhen ?thenWhenCondition .
  ?r rule:perform ?action .
}
```

which can then be used by the algorithm to be evaluated.

Recursive conditions Whenever a rule returns a value, this value can be seen as a new individual, so the whole rule can be seen as a concept. Using rules within a condition (in other words: using rules recursively) gives more expressiveness in a lot of cases.

The usage of recursion also causes some difficulties. For example, whenever you have two rules r_1 and r_2, and you use rule r_2 in a predicate of rule r_1, and the other way around, the evaluation will never stop, leading to an infinite loop. This can be prevented in multiple ways:

1. Keeping track of what rules were used, and then never calling the same rule twice.

2. Only allow the usage of rules in predicates if that rule does not have a predicate with another rule in it. (In other words: no recursion that goes deeper than one)

5.4.2 Action execution

Within the algorithm actions are only performed whenever the conditions hold. So the number of actions executed is at maximum the amount of rules that are executed. As said, actions can perform multiple things, queries on individuals, property changes on individuals or simply returning a value. An action is defined as a function with five parameters. So, the action function call looks like: $a(i_1, i_2, i_3, i_d, t)$.

Query actions Query actions are the actions that give the user information, how this information is given depends on the third parameter of the function, this is the method the information is shown to the user, described as an individual. So, if the individual is the notification center on a mobile device, that is used to give the user the result of the query.

The query itself can use the first and second parameter of the query function to display the information needed. Similar to in section 5.4.1, querying API concepts requires the usage of an API call to get the necessary information.

Property changing actions As said, actions that can change properties of individuals are able to control real-world concepts. First a query is performed to find the property of an individual, which in this case is likely another individual, that describes how to change the property of the real-world concept (using an API for example). Then this API is called to update the value defined in the rule.

The action function call also has a parameter t which tells something about the type of action it is, the type tells something about when and how often it is performed. The type usually depends on the implementation, however, some examples are:

Action type	Description
State	The action is performed continuously, whenever the conditions hold, it is performed.
At start	The action is only performed at the start, a condition can hold over a period of time (think of walking), this action is then performed when you just started walking.
At end	This is the same as "At start", with the difference that the action is only performed at the end of the period of time.

5.5 Rule and preference modeling

As almost everything is defined using rules, it is essential to get the translation from, what the user wants and what the rules do, right. By breaking down the examples given in section 5.2, it becomes visible that there are certain steps required to translate these rules from a normal sentence.

When looking at "I do not want to be disturbed when I am in a meeting." you can split this up in a subject and a predicate:

Subject	Predicate
I	do not want to be disturbed when I am in a meeting.

So the subject is what (or who) the sentence is about, the predicate tells something about the subject. After this the predicate can be split up into two parts, the condition and the event.

Condition	Event
subject in a meeting	subject does not want to be disturbed

Now if this would be translated to a rule, with the condition "subject in a meeting" the subject would have to tell somehow that he/she is in a meeting. The subject could do that himself/herself, but this would not be

done automatically then, so when defining the rule condition, it is required to think about where that information can be found. In the case of this rule it can be the calendar for the subject, thus making the subject of the sentence more specific.

For the event the same translation goes, depending on the interpretation of "subject does not want to be disturbed" this can be translated to "put my phone on silent".

This results in two subjects, one for the condition and one for the event:

Subject condition	Subject event
my planning	my phone

From here it is possible to translate this directly to the rule as seen in section 5.3.1:

1. `when` *my planning* *in a meeting*
2. ε
3. `then perform` *on silent* `using` (*my calendar*, ε, *my phone*, *my phone*, *state*)

The actual sentence is translated from "I do not want to be disturbed when I am in a meeting." to "I want my phone to be silent whenever my planning says I am in a meeting". The only relation to the subject "I" from the beginning are the individuals "my planning" and "my phone" ("my planning" and "my phone" are properties of "I"). The general steps to use to translate from a sentence to a rule are displayed in figure 8 on 41.

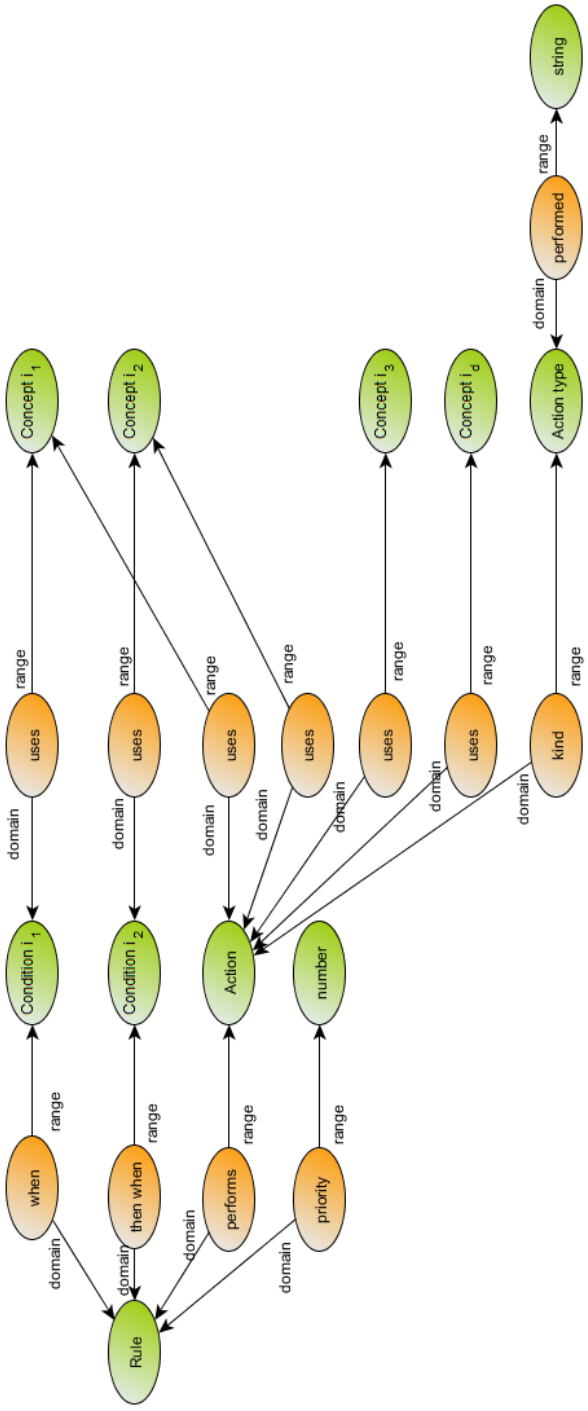

Figure 6: Rule definition in RDF (intension)

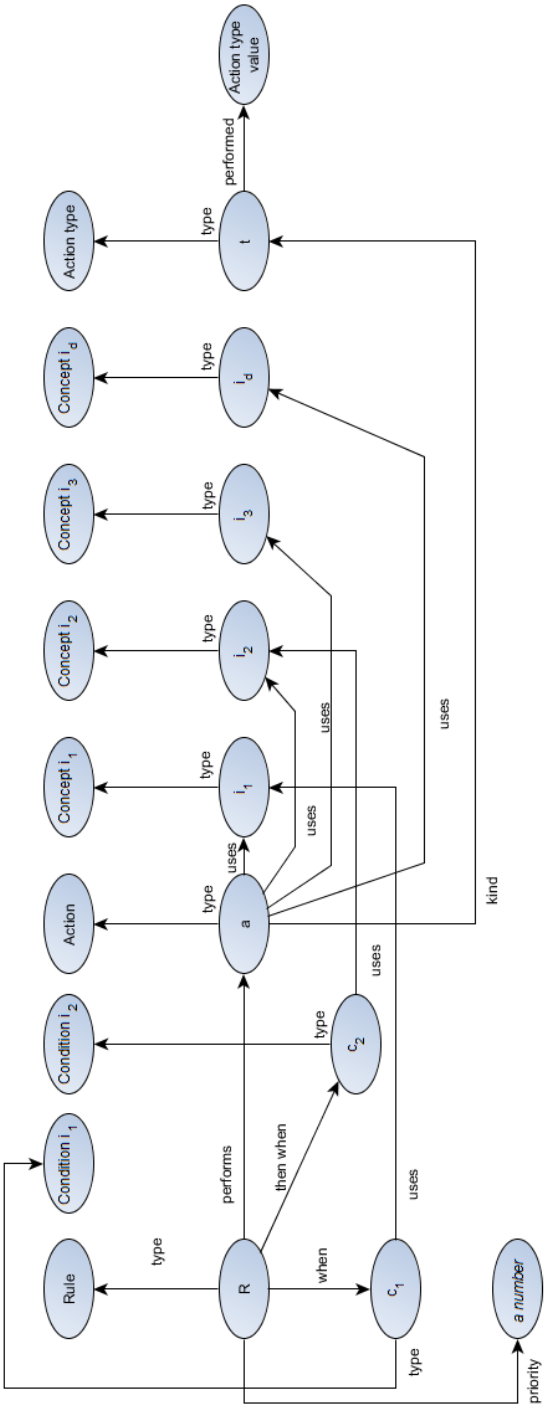

Figure 7: Rule definition in RDF (extension)

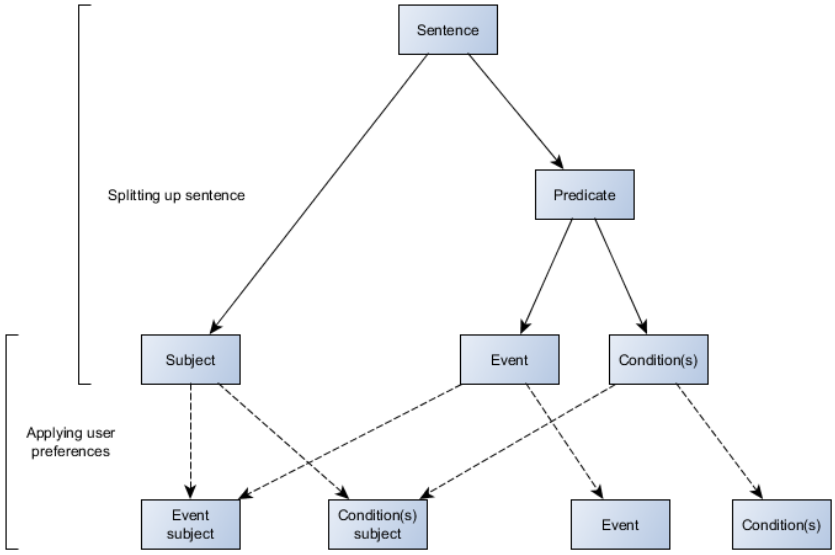

Figure 8: From sentence to rule

6 Application

The algorithm presented in section 5 can be applied for many practical applications. The approach on how to apply the algorithm to a practical application will be described in this chapter, for which the algorithm will be used to create a personal assistant.

6.1 Introduction

Creating a personal assistant for the algorithm used in this paper is a practical application that is an obvious choice for the simple reason: a personal assistant automates things in your life which are convenient to have. In section 5.2 the examples for this were:

1. Do I do enough to keep my condition up?
2. I want to be at work at 0900.
3. Is my bus delayed?
4. Make sure the heater is turned down when nobody is home.
5. An email should be read within one hour of arrival.
6. I do not want to be disturbed when I am in a meeting.

These are all helpful to have and can all be summarized under the category "personal assistant", because nearly always these rules are related to a person, a smartphone is the most obvious tool to use the algorithm on. This is because a smartphone can tell a lot about that person using its sensors, as it is the person that is nearly always carrying the smartphone on him.

Creating an implementation for a personal assistant for the algorithm described in section 5.4 a few considerations have to be made on who or what will play which role within the system. For the algorithm to reach its full potential it has to run in an environment where it can communicate with multiple other individuals, for the algorithm to work as a personal assistant application a smartphone environment is a good choice.

The system will need to be able to define rules (section 5.3.1), have a monitoring ability in some way (section 5.3.3) and perform actions to automate and control other interactive individuals (section 5.3.2).

Rules An interface can be designed for a smartphone application where you can define the rules in the predefined format. It would require knowledge of whichever individuals, a smartphone can interact with, so the ability to select these individuals becomes possible. When looking at the format of the rule (section 5.3.1), you can easily create a smartphone interface where you can select an individual from a list for the first and second part of the rule. Then a predicate for these individuals can be created which give the condition on what the state of an individual should be.

For each rule an action has also to be defined, this can again be done by creating an interface where you can select an individual and then the property that has to be changed, including the value it should change to, or the property that has to be displayed.

Automation Automation for a personal assistant is the most important part as it defines the way it assists you in certain tasks.

Distribution The distribution for this application will be centralized, as said, the smartphone is a great candidate to fulfill the role of controlling all of the concepts.

6.2 Rules

For each of the examples we want to define we can translate them to rules using the method defined in section 5.5. Where I first split up my sentence is the relevant parts, then apply my personal translation so it fits in our rule syntax.

6.2.1 Rule 1

Rule Do I do enough to keep my condition up?

Subject	Do *I* do enough to keep my condition up?
Condition	Do I *do enough to keep my condition up*?
Event	Notification with the information that answers this question

Translation First, the translation of the condition. Using the resources we have and knowing I go for a run each day, I would want to know if I run more than 30 minutes each day (when 30 minutes is the recommended norm of exercise a day needed). However, before I go for a run I do not want to be alerted that I did not fulfill the exercise requirement for that day, so as an additional condition I want the action in this rule only to be performed at 2100. This extended description gives us the following translation:

- i_1: "at 2100", translated to the individual representing the current time
- c_1: "at 2100", translated to the condition "2100", denoting the current time should be 2100
- i_2: "I run", translated to the individual "my running"
- c_2: "more than 30 minutes each day", translated to the condition "less than 30 minutes today", whenever the c_1 holds I know it has been a day, so I want to be alerted when I did not meet the required exercise time, which is less than 30 minutes that day.

Second, the action has to be translated, which should be the answer to the question asked in the rule. A general overview about the amount of exercise can be given for that day. This extended description gives us the following translation:

- a: "notification about exercise today", using i_2 it is possible to derive how much time I spent running today
- i_3: ε, no additional individual is required
- i_d: "my phone", the notification can be shown on my phone
- t: it does not matter if the action for this rule is performed at the start of 2100, during 2100 or at the end of 2100, it is just one moment in time

Definition This leads to the following definition:

1. `when` *time 2100*
2. `then when` *my running less than 30 minutes today*
3. `then perform` *notification about exercise today* `using` (*time*, *my running*, ε, *my phone*, *at start*)

6.2.2 Rule 2

Rule I want to be at work at 0900.

Subject	*I* want to be at work at 0900
Condition	I want to be at work *at 0900*
Event	I *want to be at work* at 0900

Translation First, the translation of the condition. At 0900, I want to be at work, this means that whenever I am not yet at work, I want to be notified that I should go to work. So, at a time before 0900, when my location is not at work and I have to travel a certain time to work I want to be notified in time, so I can start traveling in time. This extended description gives us the following translation:

- i_1: "my location", translated to the individual representing the current time
- c_1: "not at work", translated to the condition "2100", denoting the current time should be 2100
- i_2: "time", translated to the individual "my running"
- c_2: "current time + travel time to work = 0900", whenever the current time plus the travel time to work results in 0900

Second, the action, which can be directly translated as a notification about having to go to work. This extended description gives us the following translation:

- a: "notification about exercise today", using i_2 I can derive how much time I spent running today
- i_3: ε, no additional individual is required
- i_d: "my phone", the notification can be shown on my phone
- t: it does not matter if the action for this rule is performed at the start of 2100, during 2100 or at the end of 2100, it is just one moment in time

Definition This leads to the following definition:

1. `when` *my location* *not at work*
2. `then when` *time* *current time + travel time to work = 0900*
3. `then perform` *notification about having to go to work* `using` (*my location*, ε, ε, *my phone*, *at start*)

where the condition i_2 is a recursive condition, because I want to know what my travel time is, which is not a property of the individual "time". Making the "travel time from current location" a new rule, as follows:

- i_1: "my location", arbitrary individual to check if the same first part of the rule still holds as the first part of its parent rule
- c_1: "not at work", arbitrary condition defined for i_1
- i_2: ε, no second individual needed
- c_2: ε, no second condition needed
- a: "travel time to work", using i_3 we can return the exact time required to travel to work
- i_3: "my navigation", using a navigation API concept we can derive how much time it takes to travel to work
- i_d: ε, not needed for rules that return a value
- t: for rules that return a value, this is always a state-type

1. `when` *my location* *not at work*
2. ε
3. `then perform` *travel time to work* `using` (*my location*, ε, *my navigation*, ε, *state*)

6.2.3 Rule 3

Rule Is my bus delayed?

Subject	Is *my bus* delayed?
Condition	Is my bus *delayed*?
Event	Notification with the information that answers this question

Translation First, the translation of the condition. Whenever we translate this condition directly then the action would be performed every time the bus is delayed, this is not something we want. Because we are only interested in the bus being delayed, if we are going to travel by bus, so another implied condition is added so the we will only get this information whenever we are at a bus-stop. This extended description gives us the following translation:

- i_1: "my location", my current location
- c_1: "near bus-stop", where i_1 is near a bus-stop
- i_2: "busline 300", an API for information about busline 300
- c_2: "delayed", is the next bus delayed

Second, the action has to be translated, which should be the answer to the question asked in the rule. A notification of the amount of time that the bus is delayed, gained through the API of busline 300. This extended description gives us the following translation:

- a: "notification on how much the bus is delayed", using i_2 it is possible to get the time on how much the bus is delayed
- i_3: ε, no additional individual is required
- i_d: "my phone", the notification can be shown on my phone
- t: the action type will be "at start", so whenever the condition holds the first time the notification will be given.

Definition This leads to the following definition:

1. `when` *my location* *near bus-stop*
2. `then when` *busline 300* *delayed*
3. `then perform` *notification on how much the bus is delayed*
   `using` (*my location*, *busline 300*, ε, *my phone*, *at start*)

6.2.4 Rule 4

Rule Make sure the heater is turned down when nobody is home.

Subject	Make sure *the heater* is turned down when nobody is home.
Condition	Make sure the heater is turned down *when nobody is home*.
Event	Make sure the heater *is turned down* when nobody is home.

Translation First, the translation of the condition. This is a direct translation of what is in the rule, when nobody is home, so whenever I am not home (when living alone) the action should be performed. This extended description gives us the following translation:

- i_1: "my location", individual indicating my own location
- c_1: "not at home", whenever i_1 is not at home
- i_2: ε, no second part of the rule needed
- c_2: ε, no second part of the rule needed

Second, the action has to be translated. Which is also a direct translation of the rule: turning down the heater to a certain temperature, we just add our preference that this is 16°C. This extended description gives us the following translation:

- a: "turn heater at 16°C", using i_3 the heater can be put to 16°C
- i_3: "heater", additional individual used only by the action
- i_d: "my phone", the phone controls the heater individual
- t: "state", this is an action with a state, during the whole time the condition holds, the action should hold

Definition This leads to the following definition:

1. `when` *my location* *not at home*
2. ε
3. `then perform` *turn heater at* $16°C$ `using` (*my location*, $\underline{\varepsilon}$, *heater*, *my phone*, *state*)

Using the priority for the rules we can define a rule with a higher priority than this rule, which defines the following:

1. `when` *my location* *in a certain radius of home*
2. ε
3. `then perform` *turn heater at* $20°C$ `using` (*my location*, $\underline{\varepsilon}$, *heater*, *my phone*, *state*)

which, whenever a person is near his/her home, the heater will stay on, because this rule has a higher priority for the changes applied to the individuals. So when this person approaches home the heater is already warming up the home.

6.2.5 Rule 5

Rule An email should be read within one hour of arrival.

Subject	*An email* should be read within one hour of arrival.
Condition	An email should be read *within one hour of arrival*.
Event	An email *should be read* within one hour of arrival.

Translation First, the translation of the condition. The condition can be directly translated as the email arriving has a time-stamp, we can see whether the time is within one hour of arrival. This extended description gives us the following translation:

- i_1: "my mailbox", the mailbox for which we want to track our mails
- c_1: "receives email", whenever i_1 receives an email
- i_2: "received email", the received email which we want to have read within one hour
- c_2: "unread", whenever i_2 is unread

Second, the action has to be translated, where in this case it is not immediately clear what action the rule defines. A device like a smartphone cannot force somebody to read their email, however it can keep insisting the user to read it. This makes our action a sticky notification (a notification which cannot be removed), which forces the user to read the email or look at this notification all the time. This extended description gives us the following translation:

- a: "sticky notification that the user should read the email", using i_2 it is possible to derive if the email is read and if it is read, the notification can be removed
- i_3: ε, no additional individual is required
- i_d: "my phone", the notification can be shown on my phone
- t: "state", this is an action with a state, during the whole time the conditions hold, the action should hold

Definition This leads to the following definition:

1. `when` *my mailbox* *receives email*
2. `then when` *received email* *unread*
3. `then perform` *sticky notification that the user should read the email* `using` (*my mailbox*, *email*, ε, *my phone*, *state*)

6.2.6 Rule 6

Rule I do not want to be disturbed when I am in a meeting.

Subject	*I* do not want to be disturbed when I am in a meeting.
Condition	I do not want to be disturbed *when I am in a meeting*.
Event	I *do not want to be disturbed* when I am in a meeting.

Translation First, the translation of the condition. Which in this case it is almost a direct translation of what is defined in the rule, the only difference is that the subject is combined with the condition. This gives that my calendar (which holds my planning) should state that I am in a meeting. This extended description gives us the following translation:

- i_1: "my calendar", translated to the individual representing the current time
- c_1: "in a meeting", translated to the condition "2100", denoting the current time should be 2100
- i_2: ε, no second part of the rule is needed
- c_2: ε, no second part of the rule is needed

Second, the action has to be translated, which can be translated in the same way as the condition is translated, by combining the subject with, in this case, the event. This gives us that my phone should be put on silence whenever the conditions hold. This extended description gives us the following translation:

- a: "silent mode", using i_2 I can derive how much time I spent running today
- i_3: ε, no additional individual is required
- i_d: "my phone", the phone should be put to silent mode
- t: "state", this is an action with a state, during the whole time the condition holds, the action should hold

Definition This leads to the following definition:

1. `when` *my calendar* *in a meeting*
2. ε
3. `then perform` *silent mode* `using` (*my calendar*, $\underline{\varepsilon}$, $\underline{\varepsilon}$, *my phone*, *state*)

This (and any other) rule can also be defined in a different way, depending on your personal preferences. I could for example decide that I do not automatically want the phone set to silent mode, but actually giving me a notification at the start of a meeting, so I can put the phone to another mode manually:

1. `when` *my calendar* *in a meeting*
2. ε
3. `then perform` *notification reminding to put silent mode on* `using` (*my calendar*, $\underline{\varepsilon}$, $\underline{\varepsilon}$, *my phone*, *on start*)

7 Conclusion and future work

In this research we started off by looking at different types of data to give an idea of which data types are useful to have when looking at the interaction of a user with electronic devices. Using these data types we can find different types of concepts which are modeled using RDF, resulting in a model of a real-world concept which can be used efficiently by computer algorithms. The different types of concepts are: semantic search concepts (for which RDF was made), machine learning concepts, for which sensory data is used to describe them instead of text and API concepts, which represent concepts that have an API to communicate with them.

To make use of these RDF models a method has been presented that is able to reason with the concepts used in these models. This reasoning with concepts is done by letting a user define a set of rules. For which each rule defines two conditions and an action, whenever the two conditions hold, the action can be performed. An action either changes a property of a concept (or an individual of a concept) or looks up information of a concept. Whenever the algorithm, which processes these rules, is performed in real-time, it can perform the tasks within the rules whenever these conditions hold in real-time.

Because the rules are a translation of what the user wants, the evaluation of these rules causes them to help the user by automating some of the tasks he wants to have performed. Because the rules are a formal method of reasoning with concepts, the reasoning with concepts can automatically help the user in his/her daily life.

Now that it is possible to have an application automatically help a person in his/her daily activities some subjects can be considered for future research.

Privacy Data collection is needed to detect a lot of the concepts related to a person (see section 4.3.2) it is important to keep this data secure, because a lot of details about that person's life can be derived from that data, as shown in this research. For future research, logging and accessing this data using a secure method has to be looked at to prevent unwanted details about somebody to be leaked to the wrong people.

Automated rules For now, all of the rules have to be specified by the user, but since a lot of data about the life of a person is collected some habits may automatically be derived. These habits might lead to the creation of rules, or more specificity to them. For example, when you have a rule which

contains the individual "at work", which is usually bound to a location to know if you are at work. Being at work is done in a repeated pattern depending on the job you have, which can lead to the individual "at work" having an additional property called which describes how many hours you work. Using this property we can then reliably give navigation information to your home whenever you are at work for the suggested number of hours.

This method of automatically defining rules might save time in translating the rules for each new user using the algorithm presented in this research, as it can simply suggest some rules that somebody is likely to want to have.

References

[1] M. Blum, A. Pentland, and G. Troster, "Insense: Interest-based life logging," *MultiMedia, IEEE*, vol. 13, no. 4, pp. 40–48, 2006.

[2] L. Liao, "Location-based activity recognition," Ph.D. dissertation, University of Washington, 2006.

[3] S. Zhang, P. McCullagh, C. Nugent, and H. Zheng, "Activity monitoring using a smart phone's accelerometer with hierarchical classification," in *Intelligent Environments (IE), 2010 Sixth International Conference on*. IEEE, 2010, pp. 158–163.

[4] L. Bedogni, M. Di Felice, and L. Bononi, "By train or by car? detecting the user's motion type through smartphone sensors data," in *Wireless Days (WD), 2012 IFIP*. IEEE, 2012, pp. 1–6.

[5] J. Chon and H. Cha, "Lifemap: A smartphone-based context provider for location-based services," *Pervasive Computing, IEEE*, vol. 10, no. 2, pp. 58–67, 2011.

[6] M. Han, Y.-K. Lee, S. Lee *et al.*, "Comprehensive context recognizer based on multimodal sensors in a smartphone," *Sensors*, vol. 12, no. 9, pp. 12 588–12 605, 2012.

[7] M. Keally, G. Zhou, G. Xing, J. Wu, and A. Pyles, "Pbn: towards practical activity recognition using smartphone-based body sensor networks," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*. ACM, 2011, pp. 246–259.

[8] P. Belimpasakis, K. Roimela, and Y. You, "Experience explorer: a life-logging platform based on mobile context collection," in *Next Generation Mobile Applications, Services and Technologies, 2009. NGMAST'09. Third International Conference on*. IEEE, 2009, pp. 77–82.

[9] A. Papliatseyeu and O. Mayora, "Mobile habits: Inferring and predicting user activities with a location-aware smartphone," in *3rd Symposium of Ubiquitous Computing and Ambient Intelligence 2008*. Springer, 2009, pp. 343–352.

[10] E. Mitchell, D. Monaghan, and N. E. O'Connor, "Classification of sporting activities using smartphone accelerometers," *Sensors*, vol. 13, no. 4, pp. 5317–5337, 2013.

[11] A. M. Khan, Y.-K. Lee, S. Lee, and T.-S. Kim, "Human activity recognition via an accelerometer-enabled-smartphone using kernel discriminant analysis," in *Future Information Technology (FutureTech), 2010 5th International Conference on*. IEEE, 2010, pp. 1–6.

[12] G. M. Weiss and J. W. Lockhart, "The impact of personalization on smartphone-based activity recognition," in *Proc. of the Activity Context Representation Workshop*, 2012.

[13] KebabApps, "KebabApps," http://kebabapps.blogspot.nl/, 2014, [Online; accessed 4-April-2014].

[14] C. Apps, "Crafty Apps," http://tasker.dinglisch.net/, 2014, [Online; accessed 4-April-2014].

[15] S. OS, "Sense OS," http://www.commonsense-dashboard.com/, 2014, [Online; accessed 4-April-2014].

[16] R. Guha, R. McCool, and E. Miller, "Semantic search," in *Proceedings of the 12th international conference on World Wide Web*. ACM, 2003, pp. 700–709.

[17] M. L. Richard Cyganiak, David Wood, "Rdf 1.1 concepts and abstract syntax," W3C, Tech. Rep., 2004. [Online]. Available: http://www.w3.org/TR/rdf11-concepts/

[18] G. S. Mike Dean, "Owl web ontology language reference," W3C, Tech. Rep., 2004. [Online]. Available: http://www.w3.org/TR/owl-ref/

[19] A. S. Eric Prud'hommeaux, "Sparql query language for rdf," W3C, Tech. Rep., 2008. [Online]. Available: http://www.w3.org/TR/rdf-sparql-query/

A OWL Definition

In the OWL reference [18] the following definition can be found.

A.1 Features

- Class: A class defines a group of individuals that belong together because they share some properties. For example, Deborah and Frank are both members of the class Person. Classes can be organized in a specialization hierarchy using subClassOf. There is a built-in most general class named Thing that is the class of all individuals and is a superclass of all OWL classes. There is also a built-in most specific class named Nothing that is the class that has no instances and a subclass of all OWL classes.

- rdfs:subClassOf: Class hierarchies may be created by making one or more statements that a class is a subclass of another class. For example, the class Person could be stated to be a subclass of the class Mammal. From this a reasoner can deduce that if an individual is a Person, then it is also a Mammal.

- rdf:Property: Properties can be used to state relationships between individuals or from individuals to data values. Examples of properties include hasChild, hasRelative, hasSibling, and hasAge. The first three can be used to relate an instance of a class Person to another instance of the class Person (and are thus occurences of ObjectProperty), and the last (hasAge) can be used to relate an instance of the class Person to an instance of the datatype Integer (and is thus an occurence of DatatypeProperty). Both owl:ObjectProperty and owl:DatatypeProperty are subclasses of the RDF class rdf:Property.

- rdfs:subPropertyOf: Property hierarchies may be created by making one or more statements that a property is a subproperty of one or more other properties. For example, hasSibling may be stated to be a subproperty of hasRelative. From this a reasoner can deduce that if an individual is related to another by the hasSibling property, then it is also related to the other by the hasRelative property.

- rdfs:domain: A domain of a property limits the individuals to which the property can be applied. If a property relates an individual to another individual, and the property has a class as one of its domains,

then the individual must belong to the class. For example, the property hasChild may be stated to have the domain of Mammal. From this a reasoner can deduce that if Frank hasChild Anna, then Frank must be a Mammal. Note that rdfs:domain is called a global restriction since the restriction is stated on the property and not just on the property when it is associated with a particular class. See the discussion below on property restrictions for more information.

- rdfs:range: The range of a property limits the individuals that the property may have as its value. If a property relates an individual to another individual, and the property has a class as its range, then the other individual must belong to the range class. For example, the property hasChild may be stated to have the range of Mammal. From this a reasoner can deduce that if Louise is related to Deborah by the hasChild property, (i.e., Deborah is the child of Louise), then Deborah is a Mammal. Range is also a global restriction as is domain above. Again, see the discussion below on local restrictions (e.g. AllValuesFrom) for more information.

- Individual : Individuals are instances of classes, and properties may be used to relate one individual to another. For example, an individual named Deborah may be described as an instance of the class Person and the property hasEmployer may be used to relate the individual Deborah to the individual StanfordUniversity.

A.2 Equality or inequality

- equivalentClass : Two classes may be stated to be equivalent. Equivalent classes have the same instances. Equality can be used to create synonymous classes. For example, Car can be stated to be equivalentClass to Automobile. From this a reasoner can deduce that any individual that is an instance of Car is also an instance of Automobile and vice versa.

- equivalentProperty: Two properties may be stated to be equivalent. Equivalent properties relate one individual to the same set of other individuals. Equality may be used to create synonymous properties. For example, hasLeader may be stated to be the equivalentProperty to hasHead. From this a reasoner can deduce that if X is related to Y by the property hasLeader, X is also related to Y by the property

hasHead and vice versa. A reasoner can also deduce that hasLeader is a subproperty of hasHead and hasHead is a subProperty of hasLeader.

- sameAs: Two individuals may be stated to be the same. These constructs may be used to create a number of different names that refer to the same individual. For example, the individual Deborah may be stated to be the same individual as DeborahMcGuinness.

- differentFrom: An individual may be stated to be different from other individuals. For example, the individual Frank may be stated to be different from the individuals Deborah and Jim. Thus, if the individuals Frank and Deborah are both values for a property that is stated to be functional (thus the property has at most one value), then there is a contradiction. Explicitly stating that individuals are different can be important in when using languages such as OWL (and RDF) that do not assume that individuals have one and only one name. For example, with no additional information, a reasoner will not deduce that Frank and Deborah refer to distinct individuals.

- AllDifferent: A number of individuals may be stated to be mutually distinct in one AllDifferent statement. For example, Frank, Deborah, and Jim could be stated to be mutually distinct using the AllDifferent construct. Unlike the differentFrom statement above, this would also enforce that Jim and Deborah are distinct (not just that Frank is distinct from Deborah and Frank is distinct from Jim). The AllDifferent construct is particularly useful when there are sets of distinct objects and when modelers are interested in enforcing the unique names assumption within those sets of objects. It is used in conjunction with distinctMembers to state that all members of a list are distinct and pairwise disjoint.

A.3 Property characteristics

- inverseOf: One property may be stated to be the inverse of another property. If the property P1 is stated to be the inverse of the property P2, then if X is related to Y by the P2 property, then Y is related to X by the P1 property. For example, if hasChild is the inverse of hasParent and Deborah hasParent Louise, then a reasoner can deduce that Louise hasChild Deborah.

- TransitiveProperty: Properties may be stated to be transitive. If a property is transitive, then if the pair (x,y) is an instance of the tran-

sitive property P, and the pair (y,z) is an instance of P, then the pair (x,z) is also an instance of P. For example, if ancestor is stated to be transitive, and if Sara is an ancestor of Louise (i.e., (Sara,Louise) is an instance of the property ancestor) and Louise is an ancestor of Deborah (i.e., (Louise,Deborah) is an instance of the property ancestor), then a reasoner can deduce that Sara is an ancestor of Deborah (i.e., (Sara,Deborah) is an instance of the property ancestor).

- SymmetricProperty: Properties may be stated to be symmetric. If a property is symmetric, then if the pair (x,y) is an instance of the symmetric property P, then the pair (y,x) is also an instance of P. For example, friend may be stated to be a symmetric property. Then a reasoner that is given that Frank is a friend of Deborah can deduce that Deborah is a friend of Frank.

- FunctionalProperty : Properties may be stated to have a unique value. If a property is a FunctionalProperty, then it has no more than one value for each individual (it may have no values for an individual). This characteristic has been referred to as having a unique property. FunctionalProperty is shorthand for stating that the property's minimum cardinality is zero and its maximum cardinality is 1. For example, hasPrimaryEmployer may be stated to be a FunctionalProperty. From this a reasoner may deduce that no individual may have more than one primary employer. This does not imply that every Person must have at least one primary employer however.

- InverseFunctionalProperty: Properties may be stated to be inverse functional. If a property is inverse functional then the inverse of the property is functional. Thus the inverse of the property has at most one value for each individual. This characteristic has also been referred to as an unambiguous property. For example, hasUSSocialSecurityNumber (a unique identifier for United States residents) may be stated to be inverse functional (or unambiguous). The inverse of this property (which may be referred to as isTheSocialSecurityNumberFor) has at most one value for any individual in the class of social security numbers. Thus any one person's social security number is the only value for their isTheSocialSecurityNumberFor property. From this a reasoner can deduce that no two different individual instances of Person have the identical US Social Security Number. Also, a reasoner can deduce that if two instances of Person have the same social security number, then those two instances refer to the same individual.

A.4 Property restrictions

- allValuesFrom: The restriction allValuesFrom is stated on a property with respect to a class. It means that this property on this particular class has a local range restriction associated with it. Thus if an instance of the class is related by the property to a second individual, then the second individual can be inferred to be an instance of the local range restriction class. For example, the class Person may have a property called hasDaughter restricted to have allValuesFrom the class Woman. This means that if an individual person Louise is related by the property hasDaughter to the individual Deborah, then from this a reasoner can deduce that Deborah is an instance of the class Woman. This restriction allows the property hasDaughter to be used with other classes, such as the class Cat, and have an appropriate value restriction associated with the use of the property on that class. In this case, hasDaughter would have the local range restriction of Cat when associated with the class Cat and would have the local range restriction Person when associated with the class Person. Note that a reasoner can not deduce from an allValuesFrom restriction alone that there actually is at least one value for the property.

- someValuesFrom: The restriction someValuesFrom is stated on a property with respect to a class. A particular class may have a restriction on a property that at least one value for that property is of a certain type. For example, the class SemanticWebPaper may have a someValuesFrom restriction on the hasKeyword property that states that some value for the hasKeyword property should be an instance of the class SemanticWebTopic. This allows for the option of having multiple keywords and as long as one or more is an instance of the class SemanticWebTopic, then the paper would be consistent with the someValuesFrom restriction. Unlike allValuesFrom, someValuesFrom does not restrict all the values of the property to be instances of the same class. If myPaper is an instance of the SemanticWebPaper class, then myPaper is related by the hasKeyword property to at least one instance of the SemanticWebTopic class. Note that a reasoner can not deduce (as it could with allValuesFrom restrictions) that all values of hasKeyword are instances of the SemanticWebTopic class

A.5 Property cardinality

- minCardinality: Cardinality is stated on a property with respect to a particular class. If a minCardinality of 1 is stated on a property with respect to a class, then any instance of that class will be related to at least one individual by that property. This restriction is another way of saying that the property is required to have a value for all instances of the class. For example, the class Person would not have any minimum cardinality restrictions stated on a hasOffspring property since not all persons have offspring. The class Parent, however would have a minimum cardinality of 1 on the hasOffspring property. If a reasoner knows that Louise is a Person, then nothing can be deduced about a minimum cardinality for her hasOffspring property. Once it is discovered that Louise is an instance of Parent, then a reasoner can deduce that Louise is related to at least one individual by the hasOffspring property. From this information alone, a reasoner can not deduce any maximum number of offspring for individual instances of the class parent. In OWL Lite the only minimum cardinalities allowed are 0 or 1. A minimum cardinality of zero on a property just states (in the absence of any more specific information) that the property is optional with respect to a class. For example, the property hasOffspring may have a minimum cardinality of zero on the class Person (while it is stated to have the more specific information of minimum cardinality of one on the class Parent).

- maxCardinality: Cardinality is stated on a property with respect to a particular class. If a maxCardinality of 1 is stated on a property with respect to a class, then any instance of that class will be related to at most one individual by that property. A maxCardinality 1 restriction is sometimes called a functional or unique property. For example, the property hasRegisteredVotingState on the class UnitedStatesCitizens may have a maximum cardinality of one (because people are only allowed to vote in only one state). From this a reasoner can deduce that individual instances of the class USCitizens may not be related to two or more distinct individuals through the hasRegisteredVotingState property. From a maximum cardinality one restriction alone, a reasoner can not deduce a minimum cardinality of 1. It may be useful to state that certain classes have no values for a particular property. For example, instances of the class UnmarriedPerson should not be related to any individuals by the property hasSpouse. This situation is repre-

sented by a maximum cardinality of zero on the hasSpouse property on the class UnmarriedPerson.

- cardinality: Cardinality is provided as a convenience when it is useful to state that a property on a class has both minCardinality 0 and maxCardinality 0 or both minCardinality 1 and maxCardinality 1. For example, the class Person has exactly one value for the property hasBirthMother. From this a reasoner can deduce that no two distinct individual instances of the class Mother may be values for the hasBirthMother property of the same person.

A.6 Class intersection

- intersectionOf: OWL Lite allows intersections of named classes and restrictions. For example, the class EmployedPerson can be described as the intersectionOf Person and EmployedThings (which could be defined as things that have a minimum cardinality of 1 on the hasEmployer property). From this a reasoner may deduce that any particular EmployedPerson has at least one employer.

A.7 OWL Full additions

- oneOf: (enumerated classes): Classes can be described by enumeration of the individuals that make up the class. The members of the class are exactly the set of enumerated individuals; no more, no less. For example, the class of daysOfTheWeek can be described by simply enumerating the individuals Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday. From this a reasoner can deduce the maximum cardinality (7) of any property that has daysOfTheWeek as its allValuesFrom restriction.

- hasValue: (property values): A property can be required to have a certain individual as a value (also sometimes referred to as property values). For example, instances of the class of dutchCitizens can be characterized as those people that have theNetherlands as a value of their nationality. (The nationality value, theNetherlands, is an instance of the class of Nationalities).

- disjointWith: Classes may be stated to be disjoint from each other. For example, Man and Woman can be stated to be disjoint classes. From this disjointWith statement, a reasoner can deduce an inconsistency

when an individual is stated to be an instance of both and similarly a reasoner can deduce that if A is an instance of Man, then A is not an instance of Woman.

- unionOf, complementOf, intersectionOf (Boolean combinations): OWL DL and OWL Full allow arbitrary Boolean combinations of classes and restrictions: unionOf, complementOf, and intersectionOf. For example, using unionOf, we can state that a class contains things that are either USCitizens or DutchCitizens. Using complementOf, we could state that children are not SeniorCitizens. (i.e. the class Children is a subclass of the complement of SeniorCitizens). Citizenship of the European Union could be described as the union of the citizenship of all member states.

- minCardinality, maxCardinality, cardinality (full cardinality): While in OWL Lite, cardinalities are restricted to at least, at most or exactly 1 or 0, full OWL allows cardinality statements for arbitrary non-negative integers. For example the class of DINKs ("Dual Income, No Kids") would restrict the cardinality of the property hasIncome to a minimum cardinality of two (while the property hasChild would have to be restricted to cardinality 0).