FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

A Weather Ontology for Predictive Control in Smart Homes

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Paul Staroch
Matrikelnummer 0425426

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Mitwirkung: Dipl.-Ing. Mario Kofler

Wien, TT.MM.2013

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

A Weather Ontology for Predictive Control in Smart Homes

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Paul Staroch
Registration Number 0425426

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Assistance: Dipl.-Ing. Mario Kofler

Vienna, TT.MM.2013 ________________ (Signature of Author) ________________ (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Paul Staroch
Graf Starhemberg-Gasse 5, 1040 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, TT.MM.2013

Paul Staroch

Acknowledgements

I want to thank everyone who has worked in the field of Semantic Web and worked on countless projects, articles and books which form the foundations that made this thesis possible at all.

I wish to acknowledge Prof. Dr. Wolfgang Kastner and Dipl.-Ing. Mario Kofler for the supervision of this thesis and the opportunity to write the thesis at the Automation Systems Group of the Institute of Computer Aided Automation at the Vienna University of Technology.

Credit goes to the Norwegian Meteorological Institute for providing an easy-to-use API for weather data under a Creative Commons License.

I am particularly grateful to everyone among my relatives, friends and acquaintances who have supported me emotionally and, most notably, kept asking me about the progress of this thesis which often motivated me to carry on the work.

Abstract

In the past few years, the idea of creating *Smart Homes* has gained popularity. A *Smart Home* possesses some kind of intelligence that allows it to take work from the its inhabitants. The overall goal is to increase the inhabitants' comfort, but energy use and costs will be reduced as well.

In the context of the projects aimed at building smart home systems, this thesis aims at constructing an *OWL* ontology for weather information, containing data about both current conditions and weather forecasts. The data described by this ontology will enable smart home system to make decisions based on current and future weather conditions.

At first, the thesis will determine in which particular ways weather data can be used within smart homes. Furthermore, possible sources for weather data will be analysed. Primary sources will be weather services that are that are accessible via Internet. Optionally, local weather stations will be able to provide further data about current weather conditions. A set of Internet-based sources for weather data will be reviewed for their suitability for use within smart homes.

Afterwards, existing ontologies will be reviewed for their structure, advantages and disadvantages in order to acquire ideas being suitable to be re-used. Several well-known approaches for building new ontologies from scratch will be discussed in detail.

The thesis will follow *METHONTOLOGY*, the best-fitting of these approaches, to build *SmartHomeWeather*, an *OWL* ontology that covers both the weather data being available and the concepts required to perform weather-related tasks within smart homes while always keeping the possibility of simple and efficient OWL *reasoning* in mind.

Eventually, *Weather Importer*, a Java application, will be developed that gathers data from weather services and local weather stations and transforms it to comply with the *SmartHomeWeather* ontology.

1-2
2 notes:

Kurzfassung

In den vergangenen Jahren hat die Idee des *Intelligenten Wohnens* an Bedeutung gewonnen. Ein intelligenter Wohnraum (*Smart Home*) besitzt verfügt über eine Art Intelligenz, die es ihm ermöglicht, seinen BewohnerInnen Tätigkeiten abzunehmen. Das Ziel ist, den BewohnerInnen
3-4 2 notes: mehr Komfort zu bieten, gleichzeitig werden aber auch Energieverbrauch und Kosten gesenkt.

Im Kontext von Projekten, welche die Entwicklung von *Smart Homes* zum Ziel haben, diese Masterarbeit eine *OWL*-Ontologie für Wetterinformation entwerfen, die Daten sowohl über die aktuelle Wetterlage als auch über Vorhersagen enthält. Die von dieser Ontologie beschriebenen
5-6 2 notes: Daten werden es *Smart Home*-System ermöglichen, Entscheidungen auf Basis der aktuellen und der zukünftigen Wetterverhältnisse zu treffen.

Zunächst wird die Masterarbeit untersuchen, auf welche Art und Weise Wetterdaten in *Smart Homes* verwendet werden können. Weiters werden mögliche Quellen für Wetterdaten analysiert. Die wichtigsten Quellen werden über das Internet abrufbare Wetterdienste sein. Optional können lokale Wetterstationen weitere Daten über die aktuelle Wetterlage zur Verfügung stellen. Eine Auswahl an über das Internet verfügbaren Quellen für Wetterdaten werden hinsichtlich ihrer
07 07.08.13 by: Mario J.. Eignung zur Verwendung in *Smart Home*-Projekt untersucht.

Anschließend werden bereits existierende Ontologien bezüglich ihrer Struktur, ihren Vorteilen und ihren Nachteilen untersucht, um Ideen zu sammeln, die wiederverwendet werden können. Einige bekannte Verfahren, um neue Ontologien von Grund auf zu erstellen, werden im Detail erläutert.

Die Arbeit wird *METHONTOLOGY*, das von diesen Verfahren am besten geeignete, verwenden, um *SmartHomeWeather* zu entwerfen, eine *OWL*-Ontologie, die sowohl die zur Verfügung stehenden Wetterdaten als auch die Konzepte abdeckt, die notwendig sind, um in *Smart Homes* wetterbezogene Aufgaben zu erledigen; gleichzeitig wird darauf geachtet, einfaches und effizientes OWL-*Reasoning* zu ermöglichen.

Schließlich wird *Weather Importer* entwickelt, eine Java-Applikation, die Daten von Wetterdiensten und lokalen Wetterstationen abruft und sie so transformiert, dass sie mit der *SmartHomeWeather*-Ontologie verwendet werden können.

Contents

CHAPTER 1

Introduction

1.1 Motivation

08
07.08.13
by: Mario J..
9-11
3 notes:

Over the last years, the idea of designing and building smart homes gained increasing popularity. A smart home can be seen as a building being equipped with some kind of intelligence which enables the building to take work from its inhabitants. For instance, a smart home can take control of the *HVAC* (heating, ventilation and air conditioning) system of a building. Instead of the inhabitants having to care about maintaining an appropriate room temperature and the supply of a sufficient amount of fresh air, the smart home itself manages these tasks automatically on its own. Moreover, a smart home may regulate indoor and outdoor illumination, control home entertainment systems or monitor the building's state for unusual activity. The list of possible applications of smart homes is endless.

12
07.08.13
by: Mario J..

According to the *Intertek Research & Testing Centre* [1] which carried out a smart home project named *DTI Smart Home Project*, a smart home is "a dwelling incorporating a communications network that connects the key electrical appliances and services, and allows them to be remotely controlled, monitored or accessed. Remotely in this context can mean both within the dwelling and from outside the dwelling" [2,3]. The overall objectives that can be targeted using that approach are diverse: For instance, smart homes may be aimed at increase their inhabitants' comfort or at reducing the overall energy consumption without any loss of comfort.

The areas served by smart home systems can be categorized into six categories: Environment (*HVAC*, water management, lighting, energy management, metering), security (alarms, motions detectors, environmental detectors), home entertainment (audio visual equipment, Internet), domestic appliances (cooking, cleaning, maintenance alerts), information and communication (phone, Internet) and health (telecare, home assistance).

In order to fulfil its purpose, a smart home requires three elements: Intelligent control, home automation products, and an internal network. Intelligent control represents a gateway for managing the systems. Home automation products are components distributed in the building that either monitors or measures physical states (i.e. *sensors*), performs certain actions (i.e. *actors*) or both. Examples for sensors are thermometers, microphones, cameras or motion detectors;

actors may be switches, dimmers, room temperature controllers or window blind actuators. All components are connected using an internal network that is used by sensors to report their measurements and by actors to receive their command inputs.

As smart homes comprise a large field of research and development, there exist several projects that are aimed at developing infrastructures for smart homes. Some of these projects are:

- **Mozer's adaptive house**: The *Adaptive House* is an approach that is based on continuous monitoring the building's inhabitants regarding what actions they take and when. Based on observations collected over a certain period of time, the *Adaptive House* tries to predict the inhabitants' behaviour in order to perform the previously monitored actions (e.g. controlling the building's illumination) automatically. [4]

13
07.08.13
by: Mario J..

- **Georgia Tech Aware Home**: The *Georgia Tech Aware Home* is centered around a predefined set of use cases that are part of various scenarios such as busy families, ageing in place, or children with special needs. Throughout the building, monitors, input interfaces etc. are placed which provide assistance to the inhabitants. The *Tech Aware Home* is a rather static approach that does not include the idea of learning from the inhabitants. [5]

- **Gator Tech Smart Home**: The *Gator Tech Smart Home* is an approach geared towards simplified evolvement of the home regarding the addition of new technologies and modifications to existing application domains. The goal is the creation of *assisted environments* that utilises available sensors and actors to improve the building's inhabitant's comfort and to take routine work away from them [6].

14
07.08.13
by: Mario J..

- **eHome**: *eHome* is a project implemented for research purposes in a two-room apartment that is equipped with actors for tasks like turning light on and off, moving curtains etc. Three user interfaces (laptop, TV set and mobile phone) are available for performing the available tasks. *eHome* is not a "smart home" in the strict sense, instead "home automation" is a better suitable term. [7]

15
07.08.13
by: Mario J..

- **House_n**: *MIT*'s *House_n* is an approach based on what its designers call "subtle reminders". By the inhabitants' current behaviour, the system tries to predict what they are about to do. Based on these predictions and measurement data from various sensors, *House_n* provides support in terms of textual hints and images which are presented on simple displays mounted on windows, doors, cupboards, etc. *House_n* intentionally abstains from the use of any actors to not patronise the its inhabitants. [8]

One of the downsides that is common among many current smart home systems is the failure to exploit their full potential [9–11]. The number of parameters such a system must take into account and the variety of processes being controlled often lead to systems with a high degree of complexity. The actions taken by the system become inexplicable while optimisations or customisations imply a huge effort and is therefore avoided. The system proves to be not as powerful or flexible as desired. Users are disappointed by bad experiences with smart home systems and decide to refrain from their use.

16
07.08.13
by: Mario J..

The idea that lead to this thesis emerged from a smart home project named *ThinkHome* [12] which is developed at the *Institute of Computer Aided Automation* at the *Vienna University of Technology* [13, 14]. *ThinkHome* proposes an approach towards a smart home system that overcomes the aforementioned problems in order to provide sustainable buildings with minimized energy consumption. A building which incorporates *ThinkHome* is ought to make understandable control decisions which maintain energy efficiency and comfort without patronising its inhabitants.

The *ThinkHome* ecosystem consists of two main components: a comprehensive knowledge base and a multi-agent system. The knowledge base contains both static data (data that is seldom changed, e.g. the structure of the building, user preferences, control rules) as well as dynamic data (data that changes frequently, e.g. the current state of the building, the current state of external influences, recent measurements from sensors, current states of actors). It is implemented using a set of ontologies which define a common data model and provide a shared vocabulary. By exploiting the reasoning capabilities provided by these ontologies, the multi-agent system is enabled to make control decisions based on either predefined rules or self-learned experience.

The use of well-established semantic web technologies for its knowledge base and field-tested methods from AI, *ThinkHome* is able to provide a both flexible and powerful infrastructure which is suitable for dealing with the complexity of smart homes in an efficient and manageable way.

Like other smart home ecosystems, *ThinkHome* is designed to process input data from various sources. One of possible sources is weather data which enables *ThinkHome*'s multi-agent system to make control decisions based on current weather conditions such as temperature, humidity, precipitation, or sunshine. Furthermore, *ThinkHome* has knowledge about future weather conditions such as upcoming sudden changes of weather which may require specific control decisions in advance.

This *Predictive Control* has been the subject of numerous articles in the recent years [15–17]; however, none of these approaches builds upon a knowledge base that is built using an ontological model.

1.2 Problem statement and goal

The aim of this thesis is the development of a data model for weather data which will be utilized by smart home systems. Apart from current weather data, this model will cover future weather data over a time range suitable for the use within a smart home. This enables smart homes to use current and future states of the weather as a source of knowledge for making control decisions. While data about the current weather state can be obtained from various sensors that are installed at the building, data about future weather states must be obtained from weather services that provide forecasts for the desired period. There is a wide range of weather services available over the Internet which can be utilized for this purpose.

Besides the data model itself, an application is to be developed which imports weather data from local weather sensors, Internet weather services, or both. To provide a reference implementation, a certain weather service is selected and utilised. The application is designed in a modular way to simplify the use of different weather services.

The data model and the weather data enable smart homes to make control decisions based on current and future weather conditions.

1.3 Methodological approach

There are several existing approaches for providing weather data to smart homes as well as several approaches for covering weather data using (*OWL*) ontologies. These approaches are analysed in order to determine whether they are suitable for being reused in the context of smart homes. However, as chapter 2 discusses, none of them meets the appropriate requirements. Thus, the approaches previously analysed are reviewed for their structure, their advantages and their disadvantages.

Furthermore, a set of weather services that are available via Internet is reviewed, regarding the type of data they provide and whether they suit the requirements of a smart home, e.g. the services must provide both current data and forecasts, the data retrieved must be machine-readable and the services' terms and conditions must allow the use for smart homes. Based on these findings and the data provided by weather sensors and Internet weather services, the data which can be provided to smart homes is identified, i.e. a set of weather properties (e.g. temperature or humidity) and the time range that will be covered. One of the services is selected that will later be used to develop a reference implementation for the import of weather data.

As none of the existing ontologies qualifies to be used for smart homes, a new *OWL* ontology is designed, always keeping its intended use and simple and efficient reasoning in mind. The development process of this ontology – which is named *SmartHomeWeather* – follows one of a set of well-known approaches for ontology development. Therefore, a set of approaches is selected and evaluated. The best suitable approach is identified.

This approach is then applied to the problem domain of weather data. The result is an ontology that satisfies requirements found at smart homes as far as possible. Furthermore, a reference implementation for the import of weather data from local services services that are available via Internet is developed. This implementation uses an object-oriented model that stores the weather data and is responsible for converting input data into an appropriate format for *SmartHomeWeather*. A modular design is preferred in order to simplify the adoption of a different weather service when necessary.

Finally, all results are critically evaluated and possible future modifications, improvements and extensions are discussed.

1.4 Outline

Apart from the introduction, this thesis is structured in the following manner:

Chapter 2 discusses existing work regarding integration of weather data into *Smart Home* systems, ontologies for weather data and ontologies that may be imported by *SmartHome-Weather*. Furthermore, *ThinkHome* is presented as an example of a smart home infrastructure that comprises an knowledge base built using ontologies. *RDF*, *RDFS* and *OWL* are the technical foundations of this thesis and are covered by this chapter as well.

28
07.08.13
by: Mario J..

Chapter 3 examines various sources for weather data and determines the scope of weather data that is relevant to smart homes. Based on these findings, the range of weather data that will be used in the ontology is settled.

In chapter 4 various well-known approaches for developing ontologies are presented. Their suitability for designing the *SmartHomeWeather* ontology is evaluated and the best-suited approach is identified. The latter is described in all details relevant in the context of this thesis.

Chapter 5 describes the process of actually designing the *SmartHomeWeather* ontology.

In chapter 6 a Java application is presented that obtains weather data from appropriate services and sensors and provides them to the *SmartHomeWeather* ontology.

Finally, chapter 7 concludes about the insights from the previous chapters, summarizes all findings and gives an outlook about possible future work.

Appendix A contains all tables and listings that are omitted from the previous chapters as they are only included for reference and completeness, but not for understanding the topics covered by the previous chapters.

Appendix B contains the glossary of terms.

CHAPTER 2

Existing work

29 07.08.13 by: Mario J..

The first part of this chapter covers the foundations the work of this thesis builds upon. It gives introductions into all relevant topics, e.g. ontologies, *RDF* and *OWL*. Furthermore, it discusses the basic principles of *ThinkHome* which serves as an example of a smart home that uses a knowledge base built upon a set of ontologies.

30 07.08.13 by: Mario J..

The second part sheds light on a selection of existing work regarding weather data in smart homes and ontologies for weather data. Additionally, some ontologies which may be reused in an ontology for weather data are discussed. The final section of this chapter reviews all ontologies that are presented and identifies elements can be used for *SmartHomeWeather*.

2.1 Foundations

31 07.08.13 by: Mario J..

This chapter presents the foundations the work in this thesis builds upon: The concept of an ontology, the *Web Ontology Language* (*OWL*) and smart homes using ontologies using the example of *ThinkHome*.

2.1.1 Ontologies

In computer science, an ontology represents knowledge as a set of concepts in a certain domain and relationships between pairs of concepts [18]. The basic elements of ontologies – concepts, properties, and relations – comprise a shared vocabulary which can be used to model a certain domain.

Each object that is mapped into an ontology is represented by an *individual* (also known as *object*). Individuals of the same type can be defined to be *instances* of *concepts* (also called *classes*). Both classes and individuals can have *attributes* that specify their characteristics and properties. Two arbitrary classes or individuals can be related to each other via a *relation*.

Furthermore, ontologies may contain *function terms* (structures formed from relations that can be used in place of terms in statements), *restrictions* (descriptions of what must be true for additional knowledge to be accepted), *rules* (statements in if-then notation that describe logical

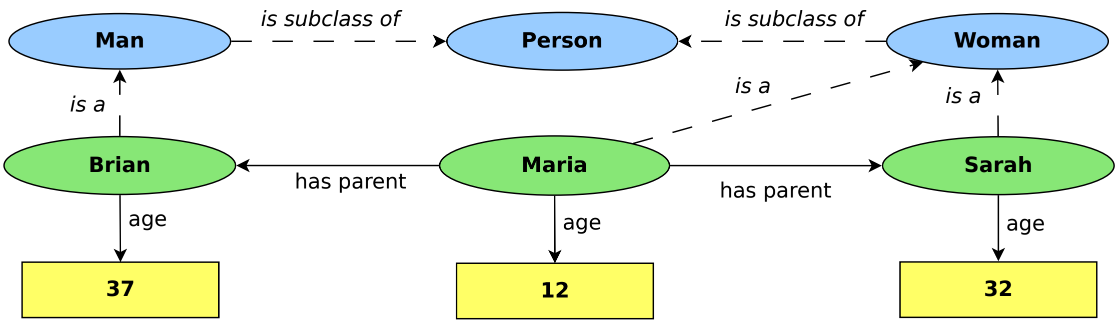

Figure 2.1: Example of a simple ontological model

inferences that can be drawn), *axioms* (core knowledge of the ontology that is known to be true), and *events* (changes to attributes or relations).

Figure 2.1 illustrates the aforementioned elements in a simple ontology: *Person*, *Man*, and *Woman* are concepts. *is subclass of* is a property that defines one concept to be a *sub-concept* of another concept (i.e. *B is subclass of A* states that every instance of *B* is also an instance of *A*). *Brian*, *Maria*, and *Sarah* are individuals; each of them is a instance of a concept which is related via the *is a* relation. *age* is a property of the individuals *Brian*, *Maria*, and *Sarah* and *has parent* is a relation which associates two individuals to each other.

32
07.08.13
by: Mario J..

This model states the following facts: Men and women are persons. Brian (age 37) is a man while Maria (age 12) and Sarah (age 32) are women. Maria has two parents, Brian and Sarah.

33
07.08.13
by: Mario J..

An important feature of an ontology is the support of automatic reasoning to deduct facts that are not explicitly stated in the data model from the given information. In order to make reasoning possible, the semantics of data models in ontologies (including *OWL*) are often based on *Description Logics* [19, 20]. These are a family of logics consisting of decidable parts of first-order predicate logic [21].

In the above example, because *Man* and *Woman* are sub-concepts of *Person*, *Brian*, *Maria*, and *Sarah* are instances of *Person*. One could define a relation *has child* that is an inverse property of *has parent*, i.e. *A has child B* if and only if *B has parent A*; then the statements *Brian has child Maria* and *Sarah has child Maria* can be deducted.

If a concept *Mother* is defined as a *Woman* who has at least one child, *Sarah* can be inferred to be an instance of this concept; the same works for an analogously defined concept *Father* with *Brian* being an instance of. Furthermore, concepts *Daughter* and *Son* can be defined (someone who has at least one parent and is a *Woman* or a *Man*, respectively). Then *Maria* can be inferred to be a *Daughter*. Additionally, properties like *has mother*, *has father*, *has son*, *has daughter* etc. can be defined.

Another core principle of an ontology is reusability [22, 23]. In order to share knowledge across various systems and to ensure interoperability of these systems, ontologies are often reused within other ontologies. Besides the simplification of knowledge exchange, ontology reuse tries to avoid duplicate work and reduces the work that is necessary to create a new ontology for a domain.

Ontologies are expressed using formal languages. There are many of these *ontology languages* such as *KIF* (*Knowledge Interchange Format*) [24], *DAML+OIL* [25] – a successor to *DAML* [26] (DARPA *Agent Markup Language*) and *OIL* (*Ontology Inference Layer*) [27] that combines features of both –, *RDFS* (*RDF Schema*) [28], and *OWL* [19]. The latter, the *Web Ontology Language*, is used in the *SmartHomeWeather* project; thus, section 2.1.2 gives a brief introduction into *OWL*.

Over the years, many ontologies have been developed. Some of them define concepts and relations that can be found across many knowledge domains. These ontologies are often imported into other ontologies in order to minimize the effort of creating new ontologies and to simplify interoperability of ontologies. Examples for such ontologies that are implemented in *OWL* or RDF *Schema* are:

- ***DOAP***: *DOAP* (*Description of a Project*) [29] is an vocabulary for describing software projects.
- **Dublin Core**: *Dublin Core* [30, 31] is a vocabulary for describing metadata of web documents, physical resources (documents, books etc.) and other objects such as works of art.
- ***FOAF***: *FOAF* (*Friend of a Friend*) [32, 33] is an ontology for describing social networks. It models persons, the relations between them, their activities and their relations to other objects.
- ***SIOC***: The *SIOC* (*Semantically-Interlinked Online Communities*) [34] is a technology built around an ontology for encoding information from Internet discussion methods (message boards, blogs, mailing lists etc.).
- ***SKOS***: *SKOS* (*Simple Knowledge Organization System*) [35, 36] is a data model for sharing and linking knowledge organisation systems (thesauri, taxonomies, classification schemes and subject heading systems).
- ***UMBEL***: *UMBEL* (*Upper Mapping and Binding Exchange Layer*) [37] is an approach towards interoperability of content on the Web. It is a vocabulary for the construction of ontologies being designed for interoperation and provides a reference structure of $25,000$ concepts that provide a scaffolding to link and interoperate datasets and domain vocabularies. Ontologies that define very general concepts which are shared between many knowledge domains are termed *upper-level ontologies*. [38]

Many standards such as *RDF*, *RDFS*, or *OWL* and some of the above ontologies have been published in the context of the W3C *Semantic Web Activity* [39] by the *World Wide Web Consortium* (*W3C*) [40]. *Semantic Web* is an approach to enrich the World Wide Web with machine-processible metadata using the technologies described in this section in order to allow better interoperability between Web pages and to ease knowledge sharing [41].

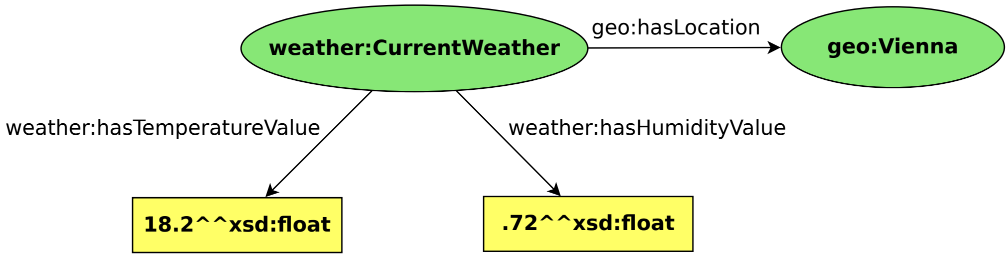

Figure 2.2: Example of a simple *RDF* model

2.1.2 OWL

The *Resource Description Framework* (*RDF*) is a standard model for knowledge representation [42]. It is specified in a set of recommendations by the *W3C*.

36
07.08.13
by: Mario J..

In *RDF*, the term *resources* is used for individuals. Each resource can have an arbitrary number of *properties*, i.e. *attributes* that associate *literate* values (e.g. numerical values, strings) to the resource or *relations* that link this resource to other resources. Resources and properties are expressed using *statements* (*triples*) which consist of three parts called *subject*, *predicate* and *object*. To identify resources and properties, *RDF* uses *URIs* (*Unified Resource Locators*)[1]. In case a resource does not have an identifier, it is a *blank node*.

37
07.08.13
by: Mario J..

Figure 2.2 depicts a simple example for a piece of knowledge from the domain of weather data expressed using *RDF*: The resource `weather:CurrentWeather` represents the current state of the weather. The property `geo:hasLocation` links the `weather:CurrentWeather` to the resource `geo:Vienna` which represents the city of Vienna, Austria; i.e. `weather:CurrentWeather` describes the weather for Vienna. `weather:weatherState` has two more properties, `weather:hasTemperatureValue` and `weather:hasHumidityValue`, which link two literal values to the resource: a temperature value of 18.2 °C and a relative humidity value of 72%. The type `xsd:float` of both literals is defined by *XML Schema* [44, 45], one of the several *XML* schema languages available that define the structure of *XML* documents.

The complete *URI* of `weather:CurrentWeather` is `http://example.org/weather#CurrentWeather` and the *URI* of `geo:Vienna` is `http://example.org/geo#Vienna`. As in *XML*, substrings at the beginning of *URIs* may be replaced by *prefixes* to avoid frequent recurrences of the same strings. The part of the *URIs* replaced by the prefix is called a *namespace* which is used to group *URIs* for elements from the same source together; e.g. all concepts, properties, and individuals defined by *SmartHomeWeather* have identifiers in the same namespace.

For the above example, the prefix `weather` has been defined to replace the string `http://example.org/weather#` and `geo` replaces `http://example.org/geo#`. This results in the identifiers `weather:CurrentWeather` and `geo:Vienna` that can be found in figure 2.2.

[1] *URIs* are strings defined by RFC *3986* [43] that uniquely identify things.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
      xmlns:geo="http://example.org/geo#"
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:weather="http://example.org/weather#">
  <rdf:Description rdf:about="http://example.org/weather#CurrentWeather">
    <weather:hasTemperatureValue
          rdf:datatype="http://www.w3.org/2001/XMLSchema#float">
      18.2
    </weather:hasTemperatureValue>
    <weather:hasHumidityValue
          rdf:datatype="http://www.w3.org/2001/XMLSchema#float">
      .72
    </weather:hasHumidityValue>
    <geo:hasLocation rdf:resource="http://example.org/geo#Vienna" />
  </rdf:Description>
</rdf:RDF>
```

Figure 2.3: *RDF* example from figure 2.2 encoded in *RDF/XML* syntax

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix weather: <http://example.org/weather#> .
@prefix geo: <http://example.org/geo#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

weather:CurrentWeather weather:hasTemperatureValue "18.2"^^xsd:float ;
    weather:hasHumidityValue ".72"^^xsd:float .

weather:CurrentWeather geo:hasLocation geo:Vienna .
```

Figure 2.4: *RDF* example from figure 2.2 encoded in *Turtle* syntax

For expressing the data that is represented by an *RDF* model, several serialization formats are available. The *RDF* recommendation is based on *RDF/XML* which maps the *RDF* model to an *XML* document [46]. The representation of the above example in *RDF/XML* can be seen in figure 2.3. As *RDF/XML* is a rather verbose format which may be difficult to read for humans, the *N3* (*Notation3*) representation for *RDF* is available which was developed with human-readability in mind [47]. *Notation3* incorporates some syntax features that go beyond the expressive power of *RDF*. A subset of *Notation3* named *Turtle* (*Terse RDF Triple Language*) is available that is limited to the features required to map *RDF* models [48]. Figure 2.4 shows the above example in *Turtle* syntax.

RDF schema (*RDFS*) is a recommendation by the *W3C* that builds upon *RDF* [28]. It introduces a set of concepts and properties adding features that go beyond the expressive power of *RDF*.

All things described by *RDF* are instances of `rdfs:Resource`. Concepts – which are introduced by *RDFS* – are instances of `rdfs:Class`, and properties are instances of `rdfs:Property`. Other concepts introduced by *RDFS* are `rdfs:Literal`, `rdfs:Datatype` and `rdfs:XMLLiteral`.

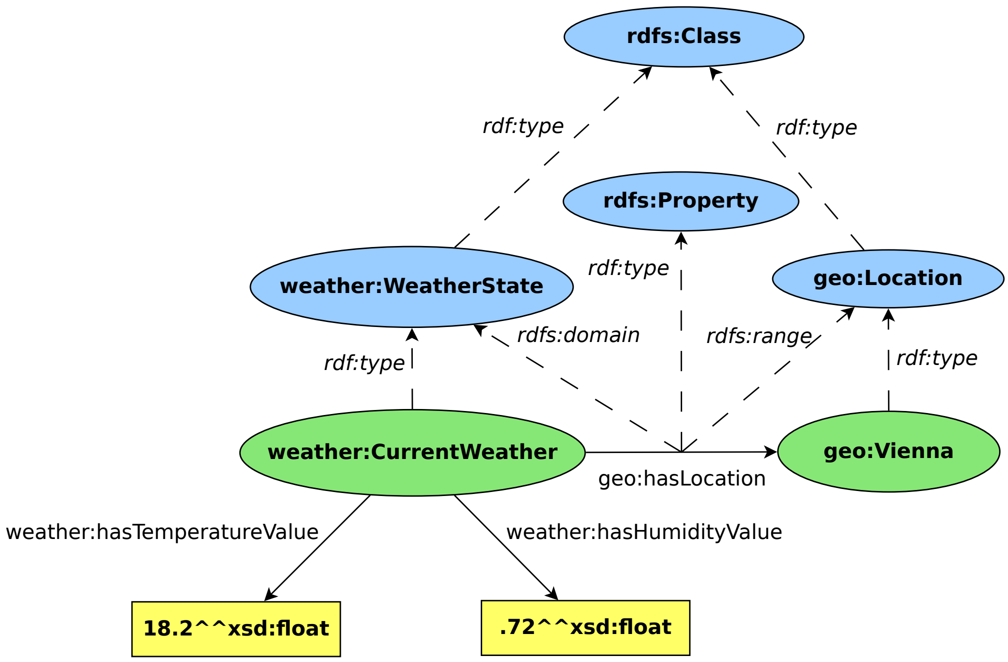

Figure 2.5: Example of a simple *RDFS* model

The property `rdfs:domain` states that any resource that has a given property is an instance of one or more classes; the property `rdfs:range` states that the value of a property is an instance of one or more classes. `rdf:type` (often abbreviated with `a`) is used to state that a resource is an instance of a class. Hierarchies of classes can be constructed using the property `rdfs:subClassOf`: `C1 rdfs:subClassOf C2` states that any instance of `C2` is also an instance of `C1`. `rdfs:subPropertyOf` is an equivalent that is used for declaring hierachies of properties. Other properties defined by *RDFS* are `rdfs:label` and `rdfs:comment`.

38 07.08.13 by: Mario J..

Figure 2.5 shows the example from figure 2.2, enriched by some elements that are introduced by *RDFS*. The data model introduces two classes, `weather:WeatherState` and `geo:Location`. `weather:CurrentWeather` and `geo:Vienna` are instances of `weather:WeatherState` and `geo:Location`, respectively. `geo:hasLocation` is a property with range `weather:WeatherState` and `geo:Location`.

39 07.08.13 by: Mario J..

RDFS comes with reasoning support [49]; e.g. in the above example, the statements

```
weather:CurrentWeather rdf:type weather:WeatherState .
geo:Vienna rdf:type geo:Location .
```

can be removed without loss of knowledge. A reasoner can deduct them from these statements:

```
weather:CurrentWeather geo:hasLocation geo:Vienna .
geo:location rdfs:domain weather:WeatherState .
geo:location rdfs:range geo:Location .
```

Many of the concepts and properties defined by *RDFS* are included in the *Web Ontology Language* (*OWL*), a more expressive ontology language than *RDFS* which is based on *RDF* and *RDFS*. *OWL* is developed by the *OWL Working Group* [50] of the *W3C*. *OWL* 1 was first published in July 2002 as a *working draft* and became a *W3C recommendation* in February 2004[2] [52]; the first *working draft* of *OWL* 2 in March 2009 and the *W3C recommendation* of *OWL* 2 in October 2009 with a second edition being finally released in December 2012 [19]. *OWL* 2 remains fully compatible to *OWL* 1, i.e. all *OWL* 1 ontologies are *OWL* 2 ontologies as well, with unchanged semantics.

Compared to *RDFS*, *OWL* introduces the following elements (among others which are omitted here):

- Properties are instances of `owl:ObjectProperty` or `owl:DatatypeProperty` (or both); the property is termed *object property* or *datatype property*, respectively. An *object property* links an individual to another individual while a *datatype property* links an individual to a literal value.

40-41 2 notes:

- The properties `owl:equivalentClass` is used to state that two classes are equivalent while `owl:allDisjointClasses` states that there is no individual that is an instance of more than one class from a set of classes.

42 07.08.13 by: Mario J..

- Similarly, the property `owl:sameAs` defines two individuals to be the same individual and `owl:differentFrom` states that two individuals can never be the same individual.

- Using the properties `owl:intersectionOf`, `owl:unionOf`, and `owl:complementOf` can be used to describe complex classes in a notation borrowed from set theory (e.g. if there are two classes, `Man` and `Woman`, the class `Person` can be defined as the class union of them).

43 07.08.13 by: Mario J..

44 07.08.13 by: Mario J..

- Using the properties `owl:allValuesFrom`, `owl:someValuesFrom`, and `owl:hasValue` classes can be defined based on the values of their properties.

- Using the properties `owl:minCardinality`, `owl:maxCardinality`, and `owl:cardinality`, the cardinality of properties per class can be limited.

- Properties can have various characteristics: A property can be inverse property of another property (`owl:inverseOf`) and two properties can be disjoint (`owl:propertyDisjointWith`). A property can be reflexive (it relates everything to itself, `owl:ReflexiveProperty`), irreflexive (no individual can be related to itself, `owl:IrreflexiveProperty`), functional (every individual can be linked to at most one other individual, `owl:FunctionalProperty`) or inverse functional (the inverse property is functional, `owl:InverseFunctionalProperty`).

[2]Both *working draft* and *recommendation* are *maturity levels* proposed by the *W3C* that indicate the state of their technical reports [51].

```
@prefix weather: <http://example.org/weather#>
@prefix geo: <http://example.org/geo#>

SELECT ?temperature
WHERE {
    ?state a weather:WeatherState .
    ?state geo:Location geo:Vienna .
    ?state weather:hasTemperatureValue ?temperature .
}
```

Figure 2.6: *SPARQL* code to query all known temperature values in the model from figure 2.2

Reasoning in *OWL* respects the *open world assumption* [53, 54]. If some statement cannot be inferred, it is not allowed to assume that this statement is false. Hence, reasoning in *OWL* is *monotonic*: Adding more information to a model cannot cause anything become false that has previously known to be true, and vice versa [55].

Queries on *RDF*, *RDFS*, and *OWL* models are often performed using *SPARQL*, a query language for *RDF* [56]. Figure 2.6 shows an example for the use of *SPARQL* to query all temperature values known for Vienna, Austria in the *RDFS* model depicted in figure 2.5.

OWL ontologies are often designed using semantic editors like *Protégé* [57]. Common reasoners for *OWL* include *Pellet* [58], *RacerPro* [59], *FaCT++* [60] and *HermiT* [61]. They all implement reasoning as specified by the *OWL* specification; implementation specific differences affect details that are not covered by the *OWL* specification. *Protégé* and *Pellet* are used to develop the *SmartHomeWeather* ontology in chapter 5.

2.1.3 ThinkHome

Section 1.1 gives a short introduction into the basic ideas of *ThinkHome* which serves as an example for an ontology-based smart home infrastructure. This section goes further into detail into the two main components that form the *ThinkHome* system: a comprehensive knowledge base and a multi-agent system.

The knowledge base is organised into six sub-categories [13, 14]: *Building* (structure of the building), *Actor* (human system users and software agents), *Comfort* (parameters that affect the users' well-being), *Energy* (energy providers), *Process* (the users' activities), and *Resource* (available equipment). Some parts of the knowledge base are static data (data that is seldom changed, e.g. the structure of the building, user preferences, control rules) while others are dynamic data (data that changes frequently, e.g. the current state of the building, the current state of external influences, recent measurements from sensors, current states of actors). The knowledge base is implemented in *OWL*. This enables the use of the reasoning capabilities offered by semantic reasoners. With *SPARQL*, a powerful query language for *OWL* models is available that is suitable for accessing *ThinkHome*'s knowledge base. The *Apache Jena Framework* [62] as well as the *Virtuoso Universal Server* [63] are environments used by *ThinkHome*.

A multi-agent system is both a software paradigm and a method supporting distributed intelligence, interaction and cooperation to work towards predefined goals [64]. *ThinkHome* incorporates a multi-agent system to implement advanced control strategies. *ThinkHome*'s *agent* *society* consists of various different agents, each serving a specific purpose. As the core of the agent system, the *Control Agent* is responsible for executing control strategies: it obtains data from various sources to get a global view of the system state, then calculates an appropriate control strategies and initiates its execution. Other agents are the *User Agent* which acts on behalf of a user to provide her with a comfortable environment; the *Global Goals Agent* which enforces global policies towards energy efficiency; the *Context Inference Agent* which sets actions in context with users, location, and time; the *Auxiliary Data Agent* which obtains data from various sources; the *KB Interface Agent* which provides the link between the knowledge base and the multi-agent system and is responsible for data exchange in all parts of the system; and the *BAS Interface Agent* which acts as an interface between the agent society and the building automation systems (*BAS*) available in the smart home.

As for the knowledge base, *ThinkHome* uses a set of well-established technologies to implement the multi-agent system, such as the *Java Agent DEvelopment Framework* [65] and the *JACK Intelligent Agents* framework [66].

2.2 Ontologies for weather data

This section presents a selection of the ontologies that have been designed to cover the domain of weather data. Possibly one of them qualifies to be used as a basis for *SmartHomeWeather*. If not, their advantages and disadvantages are analysed in order to avoid their shortcomings in *SmartHomeWeather* and benefit from their advantages.

This section covers *Semantic Sensor Web* (section 2.2.1), the SSN *ontology* (section 2.2.2), SWEET (section 2.2.3), and *NextGen* (section 2.2.4). There are several other approaches such as in the *SENSEI project* [67] which are not covered here.

Apart from ontology-based weather data models, there exist several approaches which incorporate both current and predicted weather data without embodying an ontology. A commonly used approach is the use of a mathematical model that allows the transformation of the problem of predictive control based on weather data to an optimization problem [68, 69]. As the structure of these approaches differs greatly from the ontological approach that is found at smart home systems like *ThinkHome*, these approaches are not covered here.

2.2.1 Semantic Sensor Web

Sensor Web Enablement (*SWE*) [70] is an initiative by the *Open Geospatial Consortium* (*OGC*) [71] for building networks of sensors based on Web technologies. Both sensors and archived sensors are intended to be discovered, accessed and optionally controlled using open standard protocols and interfaces. *SWE* is a suite of standards, each specifying encodings for describing sensors, sensor observations, and/or sensor interface definitions.

There exists a huge number of sensor networks around the globe comprising sensors for a large set of different phenomena. Therefore a vast amount of data is available, the need arises to structure that data and allow interoperability between different sensor networks. *Semantic Sensor Web* (*SSW*) is an approach that builds upon *SWE* and Semantic Web activities by the *W3C* which aims at annotating sensor data with semantic metadata to increase interoperability and to provide contextual information essential for situational knowledge [72]. Semantic metadata includes spatial, temporal, and thematic data.

The ontology of *SSW* – which is referred to as the SSW *ontology* from now on – is built upon seven top-level concepts, excluding the concepts *Location* and *Time* which are imported from other sources. These top-level concepts are *Feature* (an abstraction of a phenomenon from the real word, e.g. a weather event such as a blizzard), *Observation* (the act of observing a property or phenomenon with the goal of a value of a property), *ObservationCollection* (a set of *Observation*s), *Process* (a method, algorithm, or instrument, or a system of these), *PropertyType* (a characteristic of one or more types of *Feature*s), *ResultData* (an estimate of the value of some property generated by a known procedure), and *UnitOfMeasurement* (as the name suggests, a unit of measurement).

The *SSW* ontology includes temporal data using *OWL-Time* (see section 2.3.2), units of measurements and geographical data based on the *Basic (WGS84 lat/long) Vocabulary* (see section 2.3.1). The concept *WeatherObservation* comes with five predefined sub-concepts designed to map observations of atmospheric pressure, precipitation, radiation, temperature, or wind, respectively. Additional concepts may be added for observations of other phenomena.

The *SSW* ontology does not support forecast values, but extending the *SSW* ontology to implement those would be possible.

Applications of *SSW* include work on situation awareness based on the metadata specified by *SSW* [73] and an architecture for a distributed semantic sensor web [74].

The only part from the *SSW* ontology that could be reused for *SmartHomeWeather* is the concept *Observations* together with its sub-concepts; however, a number of additional sub-concepts would have to be added. The *Basic (WGS84 lat/long) Vocabulary* and *OWL-Time* can be used by the *SmartHomeWeather* ontology without the use of the *SSW* ontology. For units of measurements there are other approaches than the *SSW* approaches available that qualify for being used by *SmartHomeWeather*. Therefore, it was decided not to use the *SSW* ontology for *SmartHomeWeather*.

2.2.2 *SSN* Ontology

Another approach towards semantically enriched sensor network based on *SWE* is an OWL *2* ontology created by the W3C *Semantic Sensor Network Incubator group* (*SSN-XG*) [75] which is referred to as the SSN *Ontology* [76]. The goal of this ontology is to simplify managing, querying, and combining sensors and observation data from different sensor networks.

The *SSN* ontology uses *DOLCE-UltraLite* [77] as *upper-level ontology*. It defines 41 concepts and 39 object properties which are organised into ten modules: *ConstraintBlock* (for defining conditions on a system's or a sensor's operation), *Data* (for encoding any input from sensors), *Device* (for defining devices in the sensor network, mostly sensors), *Deployment* (for specifying the deployment of *Device*s), *MeasuringCapability* (properties of sensors, e.g. accuracy or response time), *OperatingRestriction* (for defining conditions under which the system is expected to operate, e.g. the life time of batteries or maintenance schedules), *PlatformSite* (entities to which other entities – sensors and other platforms – can be attached), *Process* (a procedure that changes the system's state in some way, takes some input and yields some output), *Skeleton* (for

53
07.08.13
by: Mario J..

mapping real-world phenomena, their properties, and their relations to sensors) and *System* (for describing parts of infrastructure, e.g. the whole network, a component, its subsystems etc.).

When used, the ontology can be used to view at the knowledge base from a number of perspectives: The *sensor perspective* (which sensors are available; what and how do they sense), the *observation perspective* (focusing on observations and related metadata), the *system perspective* (systems of sensors), and the *property perspective* (properties of physical phenomena and how they are sensed).

Work based on the *SSN* ontology includes its use for the representation of humans and personal devices as sensors [78], its application in sensing for manufacturing [79], and as part of a linked data infrastructure for *SWE* [80].

The primary reason for not using the *SSN* ontology for *SmartHomeWeather* is the fact that it was published later than the development of *SmartHomeWeather* started. Furthermore, for the purpose of *SmartHomeWeather*, the *SSN* ontology uses a level of abstraction that makes its use in smart homes too complicated. *SSN* is well engineered towards mapping sensor networks and its observations, but it does not qualify for only representing sensor data without mapping the details of a sensor network. Additionally, forecast data is currently not supported, but an appropriate extension would be possible.

2.2.3 SWEET

SWEET is a set of more than 200 ontologies comprising about 6000 concepts [81, 82]. It is developed by *NASA*'s *Jet Propulsion Laboratory* at the *California Institute of Technology* [83]. The initial version which dates back to 2003 is based on *DAML+OIL* [25, 84]. It structured its ontologies into the following categories:

- *Earth Realm*: The "spheres" (e.g. atmosphere or ocean) of Earth belong to this category.
- *Non-Living Element*: This category includes non-living building blocks on nature (e.g. particles or electromagnetic radiation).
- *Living Element*: All plants and animal species belong to this category.
- *Physical Properties*: This category contains physical properties (e.g. temperature or height).
- *Units*: This category comprises units of measurement for all literal values used in the ontologies.
- *Numerical Entity*: This category includes numerical extents (e.g. interval or point) and numerical relations (e.g. greatherThan or max).
- *Temporal Entity*: This category includes ontologies that cover the temporal domain: temporal extents (e.g. duration or season) and temporal relations (e.g. after or before).
- *Spatial Entity*: Ontologies that cover the spatial domain fall into this category: spatial extents (e.g. country or equator) and spatial relations (e.g. above or northOf).

- *Phenomena*: This category contains ontologies that define transient events; a phenomenon that is described by such an ontology crosses bounds of other ontology domains (e.g. hurricane or terrorist event).

55
07.08.13
by: Mario J..

Now, *SWEET* is now implemented in *OWL*. The current version, *SWEET* 2.3 was last updated in 2012.

Works based on *SWEET* include work on integrating volcanic and atmospheric data in the context of volcanic eruptions [85], an ontology of fractures of the Earth's crust [86], and an extension of *SWEET* by climate and forecasting terms [87].

In the context of smart homes, the *SWEET* ontologies are the best-qualified approach for reusing them in an weather ontology. Units of measurements, temporal definitions, and specifications of geographical positions are supported. However, *SWEET* comes with a few downside that finally led to the decision not to use them in *SmartHomeWeather*:

56
07.08.13
by: Mario J..

- Although *SWEET* is separated into a large number of ontologies, there are many dependencies between ontologies. Importing a single ontology into an *OWL* ontology causes the import of all of its dependencies, the import of all dependencies of all dependencies etc. It is impossible to import one ontology from *SWEET* without importing dozens of other ontologies.

- *SWEET* does not currently cover future events.

- *SWEET* does not cover all weather elements that are relevant.

Extensions to the *SWEET* ontologies in order address the second and the third issue would be possible; an extension of *SWEET* to include forecast data already exists [87]. However, as a large number of extensions would be necessary, it was decided not to use any parts of *SWEET* for *SmartHomeWeather* and to build a new ontology from scratch instead.

2.2.4 NextGen

NextGen (*Next Generation Air Transport System*) [88] is a large-scale project carried out by the *Federal Aviation Administration* (*FAA*), an organisation being responsible for all aspects of civil air traffic in the United States [89]. The goal of *NextGen* is to completely reorganise the US airspace to shorten flight paths, save time and fuel, minimise delays, increase capacity, and improve safety. One of the elements that *NextGen* consists of is *Next Generation Network Enabled Weather* (*NNEW*) which is designed to provide a comprehensive view on the weather across the country, built from thousands of single weather observations. Once completed, *NNEW* is expected to reduce weather-related delays in US airspace to the half of its current magnitude; currently, approximately 70 percent of all air traffic delays are attributable to weather [90].

In order to efficiently process huge amounts of input weather data, *NNEW* incorporates a knowledge base implemented using a ~~set~~ of ontologies. The *NNEW* ontology is centered around concepts and relations describing past, current, and future weather phenomena.

57
07.08.13
by: Mario J..

The *NNEW* ontology is built on to of the *SWEET* ontologies (see section 2.2.3) to map weather phenomena. Extensions to *SWEET* include additional weather phenomena and concepts

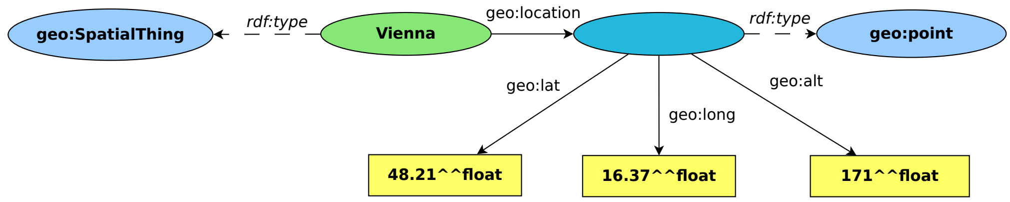

Figure 2.7: Example of the use of the *Basic Geo (WGS84 lat/long) Vocabulary* to specify the location of Vienna, Austria (N 48.21°, E 16.37°, 171 m above *MSL*); all concepts and properties in the `geo` namespace are part of this vocabulary.

and relations that lead to the *4-D Wx Data Cube* (*Four Dimensional Weather Data Cube*) which uses time as the fourth dimension for the "location" of weather observations.

58
07.08.13
by: Mario J..

It was decided to use the *NNEW* ontology for *SmartHomeWeather* for the same reasons as this decision was made in the case of *SWEET*. In the context of smart homes, the *NNEW* ontology appears just as an extension to *SWEET* and therefore it is not qualified in equal measure. Furthermore, *NNEW* is still work in progress and no final version is available that could be seen as a standard.

2.3 Related ontologies

This section discusses some ontologies that cover domains that are related to weather data, e.g. location data, temporal data, or units of measurements. These are candidates for being reused as part of *SmartHomeWeather*.

2.3.1 Location data

To handle geographical location data, the W3C *Semantic Web Interest Group* (*SWIG*) developed the *Basic Geo (WGS84 lat/long) Vocabulary* [91]. It introduces a concept called *Spatial thing* and its attributes *lat*, *long* and *alt* according to the *WGS-84 geodetic reference system* [92]. Figure 2.7 shows an example of how the vocabulary is used.

As a downside, the vocabulary contains some data properties that are incorrectly defined to be annotation properties. Thus, they must be redefined to be data properties whenever used in an *OWL* ontology.

Furthermore, the *Basic Geo (WGS84 lat/long) Vocabulary* is not a standard; is has not even been submitted to the W3C *recommendation track* for standardisation. No work on the vocabulary has been done since 2006. The W3C *Geospatial Incubator Group* proposed the introduction of *GeoRSS XML* and *Geo OWL* in 2007 [93] which are designed to enrich *RSS*/Atom feeds and *OWL* ontologies with geographical information. Although *GeoRSS* seems to have gained popularity over the last few years, *Geo OWL* continues to lead a miserable existance. The last reports by the W3C *Geospatial Incubator Group* were published in 2007 [93, 94]. No standard which may be qualified to supersede the *Basic Geo (*WGS84 *lat/long) Vocabulary* has yet been released.

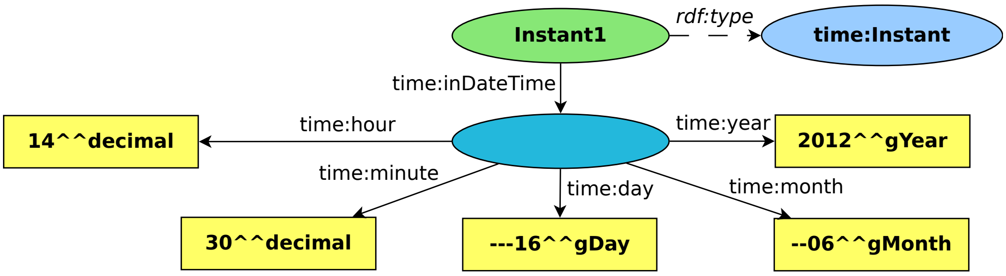

Figure 2.8: Example of the use of the *OWL-Time* for describing an instant (July 16, 2012 at 14:30).

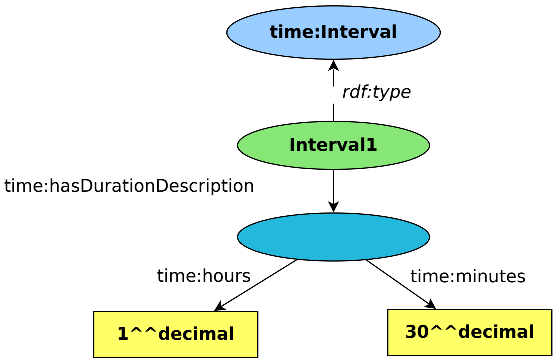

Figure 2.9: Example of the use of the *OWL-Time* for the description of the interval of 90 minutes (one hour and 30 minutes).

2.3.2 Date and time

For specifying temporal properties, the *W3C* offers a *working draft* of *OWL-Time* [95], a *Time Ontology in* OWL. It defines the concept called *Temporal entity* that can either be an *Instant* or an *Interval*. Using appropriate attributes, the properties of a *Temporal entity* are specified. Although *OWL-Time* is in the state of a *working draft* since September 2006, the main concepts and attributes that will be reused by other ontologies are likely to remain unchanged in future releases of that ontology.

Figure 2.8 shows an example of *OWL-Time* being used to specify an instant while figure 2.9 demonstrates how a time interval is specified using *OWL-Time*.

2.3.3 Units of measurements

As the use of units of measurement is a topic that occurs in many ontologies, there are several different approaches to cope with it.

The *Measurement Units Ontology* [96, 97] is a simple and light-weight approach to enrich measurement values in an ontology with appropriate units. Some of the most important units are pre-defined, others can easily be added when needed. The *Measurement Units Ontology* (*MUO*)

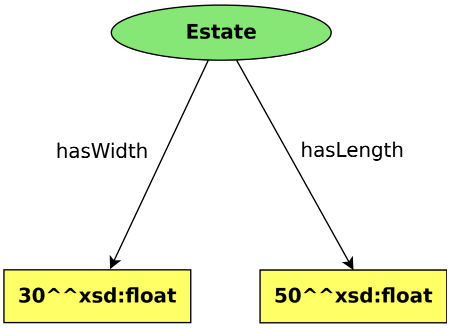

Figure 2.10: Example of a data model lacking units of measurement; the model represents an estate with a length of 50 metres and a width of 30 metres.

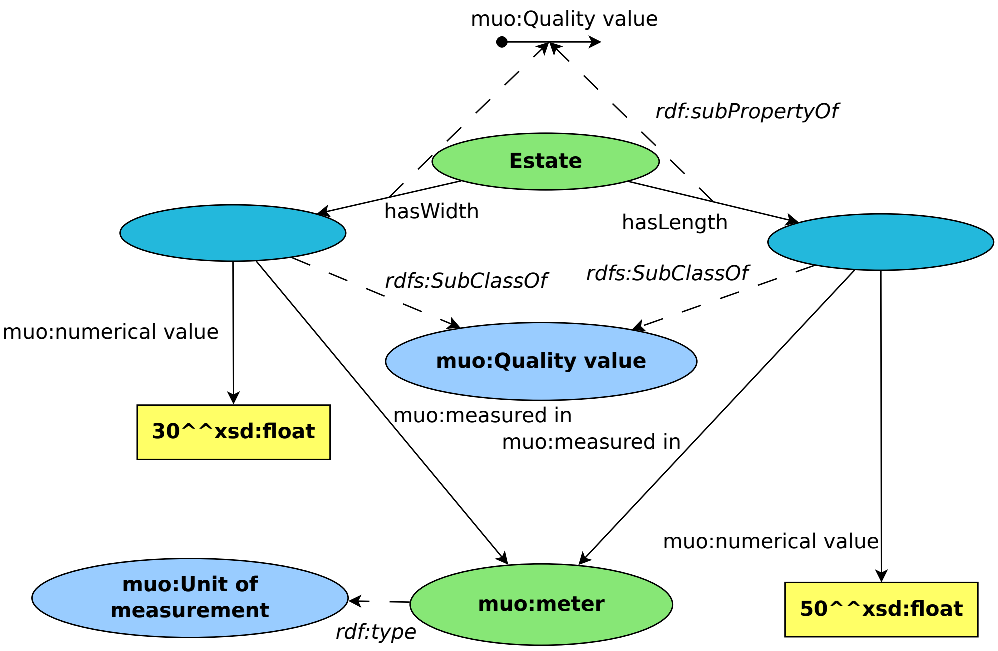

Figure 2.11: Example of the use of *MUO* for the introduction of units of measurements.

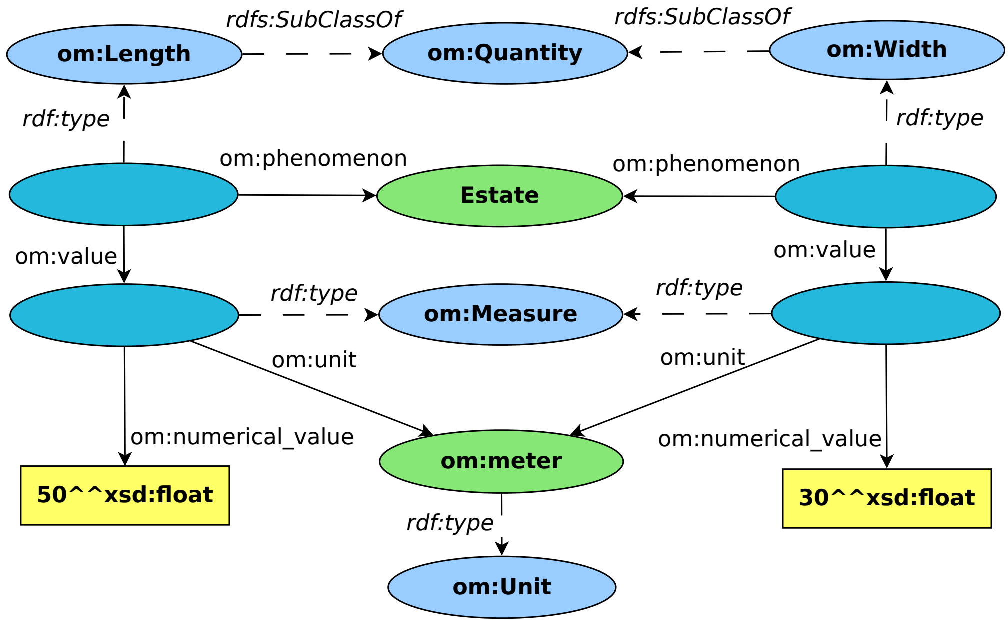

Figure 2.12: Example of the use of *OM* for the introduction of units of measurements.

59
07.08.13
by: Mario J..

is still work in progress, however it is not expected that it might change heavily in the future. Everything that will be reused by other ontologies will remain unchanged. Hence, *MUO* can be imported into any ontology without problems.

60
07.08.13
by: Mario J..

Figure 2.10 shows the representation of an estate with its length and width specified using the datatype properties `hasLength` and `hasWidth`. In figure 2.11, *MUO* is introduced to specify that both length and width are measured in metres: `hasLength` and `hasWidth` become object properties which link blank nodes to the individual `Estate`; `hasLength` and `hasWidth` are now both sub-properties of `muo:Quality value`. Each blank node has two properties, `muo:numerical value` states the literal value while `muo:measured in` gives the unit of measurement for the literal value. The unit (`meter` in this case) is represented by an instance of `muo:Unit of measurement`.

The *Ontology of Units of Measure and Related Concepts* (*OM*) [98,99] is another promising approach for adding measurement units to *OWL* that even provides features like the conversion between different units for the same quantities and representation and checking of formulas. Figure 2.12 shows an example of the usage of *OM*. In this example, the specification the length of the estate is centered around a blank node which is an instance of `om:Length` which is a subconcept of `om:Quantity`. The blank node links to `Estate` using the property `om:phenomenon` and to another blank node via the property `om:value` which is an instance of `om:Measure`. This instance has a datatype property of type `om:numerical_value` which specifies the numeric value of the estate's length, while the object property `om:unit` links to an instance of `om:Unit` named `om:meter`. The width of the estate is stated in a similar way.

OM is a rather large ontology (nearly 2.4 MiB in *RDF/XML syntax*) compared to the *SmartHomeWeather* ontology and related to the purpose it would fulfil in *SmartHomeWeather*; furthermore, at the time when the article about *OM* was published, development of *SmartHomeWeather* had already completed. Hence, *OM* was not taken into account for being used in the *SmartHomeWeather* ontology.

Besides the *Measurement Units Ontology* and the *Ontology of Units of Measure and Related Concepts*, several other ontologies have been examined, but all of them have shortcomings that render their use in the *SmartHomeWeather* ontology impossible. These ontologies are:

- ***SWEET***: Besides concepts, attributes and individuals for atmospherical phenomena, *SWEET* also comes with support for literals more precisely specified by units [84]. However, as mentioned above, *SWEET* is an ontology that is inappropriate for use in *SmartHomeWeather*.

- ***QUDT***: *QUDT*, a set of ontologies for *Quantities, Units, Dimensions and Data Types in OWL and XML*, is a promising approach for adding support for units to an *OWL* ontology [100]. However, *QUDT* does not work in *Protégé* together with the *Pellet* reasoner. Using *QUDT* would require several changes to its *OWL* files. Hence, it cannot be used in the *SmartHomeWeather* ontology.

- ***QUOMOS***: The OASIS *Quantities and Units of Measure Ontology Standard* (*QUOMOS*) [101] is a project that aims at developing "an ontology for quantities, systems of measurement units, and base dimensions for use across multiple industries". However, as no deliverables have been released at the time of writing, it cannot be used in *SmartHomeWeather*.

- ***OBO* Foundry Initiative**: The OBO *Foundry Initiative* [102] is an initiative that aims at collecting ontologies for use in the biomedical domain. The list of *The Open Biological and Biomedical Ontologies* (abbreviated by *OBO*) includes an ontology for units of measurements [103, 104]. This ontology apparently covers most of the units that are used in the *SmartHomeWeather* ontology. However, it lacks any documentation and therefore is unsuitable for being used in *SmartHomeWeather*.

2.4 Conclusion

As section 2.2 discusses, unfortunately no existing ontology covers the domain of weather data in a way suitable for using it as a starting point for *SmartHomeWeather*.

Thus, a completely new ontology is created (see chapter 5). From the insights gained in this chapter from existing weather ontologies, the following aspects are considered for the development of *SmartHomeWeather*:

- Many of the existing weather ontologies are intended to map a whole sensor network. In the case of *SmartHomeWeather*, this is not necessary. To avoid overhead, only observations of weather elements (temperature, humidity etc.) will be covered, not their sensors. One goal of *SmartHomeWeather* is to keep it as sophisticated as required, but as simple as possible.

- The *Basic Geo (WGS84 lat/long) Vocabulary* and *OWL-Time* are used by some of the existing ontologies. This qualifies these vocabularies to adequately model geographical and temporal data in a number of different domains. Hence, both vocabularies will be imported by *SmartHomeWeather*.
- Some ontologies include concepts and properties for representing units of measurement. *SmartHomeWeather* will support units of measurement as well, in fact by using the *Measurements Units Ontology*.

63
07.08.13
by: Mario J..

CHAPTER 3

Weather data

64-66 3 notes:

Based on the insights acquired from existing work in chapter 2, this chapter aims at compiling a set of data elements about the weather which are either necessary for providing useful data to smart home systems or which would add benefit to the data provided. Furthermore, possible sources are evaluated with respect to their suitability for the given context.

3.1 Weather information

In order to identify which data is required for the *SmartHomeWeather* ontology, it is necessary to define the scope that shall be covered by the ontology. When designing an ontology, requirements analysis is often centered around a set of *competency questions* [105–108]. If the ontology is able to provide answers to all of these competency questions, its requirements are met.

Below are the questions that have been identified to be adequate competency questions for *SmartHomeWeather*; the list stems from analysis of the processes at a *Smart Home* that may be influenced by weather.

- What is the current weather situation?
- What will the weather situation be in one hour, in two hours, . . . , in 24 hours?
- What is the current temperature, humidity, wind speed, . . . ?
- What will be the temperature, humidity, wind speed, . . . in one hour, in two hours, . . . , in 24 hours?
- 67 07.08.13 by: Mario J.. What will be the minimum temperature, humidity, . . . over the next 24 hours? What about maximum values?
- Will the weather change? Will the temperature, humidity, . . . rise or fall?
- Does it rain? Will it rain in the next hours? Will it rain today?

- Will there be sunshine today?
- Do we need to irrigate the garden?
- Will there be severe weather?
- Will temperature drop/stay below 0 °C?
- When can we open windows and when do we have to keep them shut?
- When do we need sun protection?
- When will it outside be colder than inside the house? When will it be warmer?

These competency questions will again be used in section 5.2 when the *Ontology requirements specification document* is created.

68-69 2 notes:

The idea of providing smart home with future weather data is to enable smart homes to prepare for upcoming weather situations. There are Internet weather services that provide forecasts for several days (see section 3.3.1). However, the further the time described by a forecast lies in the future, the more inaccurate the forecast becomes [109, 110]. Additionally, no decisions in a *Smart Home* have been identified that require the availability of a forecast about a time more than a few hours in the future. Hence, the period of 24 hours has been chosen as a compromise between the deteriorating accuracy of weather forecasts over time and the time period smart homes require weather data for.

The above competency questions can be answered when the (predicted) state of the weather for particular points of time is known. The state of the weather is given by measurement values of certain *weather elements*. These weather elements are temperature, relative and absolute humidity, dew point temperature, wind speed, wind direction, precipitation, cloud coverage and others. [111]

A set of measurement value for one or more points of time is called *weather state* from now on.

70-71 2 notes:

Apart from weather elements, Internet weather services often provide some information that will be called *weather condition* from now on. Generally speaking, the weather condition is what someone tells you if you ask her about the current weather situation. Examples for the overall weather condition are “Sun”, “Rain”, and “Fog”. Some weather conditions can be split up into several conditions, e.g. “It is overcast and raining” into “Overcast” and “Rain”.

Sources for weather data are weather sensors and weather forecasts. As smart homes perform no weather forecasting on its own, forecast data is gathered from weather services via Internet. While data about the current weather state can be obtained from both weather sensors and Internet services, the only source for data about future weather states are Internet services.

72 07.08.13 by: Mario J..

However, there there are weather elements no sensor can provide data about, and there are weather elements that are not available from any of the Internet services presented in section 3.3. Section 3.2 covers data that can be obtained from weather sensors; section 3.3 discusses weather data that can be fetched from Internet services. Based on the findings in these two sections, 3.4 presents a set of weather elements that are incorporated into *SmartHomeWeather*.

3.2 Sensor data

While there are stand-alone solutions for fetching weather data from sensors, in many *home automation* systems *fieldbus* systems are used.

3.2.1 *Fieldbus* systems

Over the past few years, several of *fieldbus* systems have emerged in the context of *home automation*. A *fieldbus* is a network system for real-time distributed control [112], standardised as *IEC 61158* [113]. A *Programmable Logic Controller* (*PLC*), a computer which is designed specifically for automation purposes, can utilize a *fieldbus* to retrieve sensor data from *sensors* and send control commands to *actuators*. Among the sensors that are available for *fieldbus* systems, there exist sensors for a wide variety of weather elements such as temperature or humidity.

There are several competing standards for *fieldbus* systems, including:

- **Bitbus:** *Bitbus*, created in 1983 by *Intel Corporation*, is the oldest *fieldbus*. In 1991, it was approved as an international standard as *IEEE 1118* [114]. It employs *twisted pair* cabling for connecting its components.

- **KNX:** *KNX* is an international standard (*ISO/IEC 14543-3* [115,116]) which is the successor to three previously used *fieldbus* systems, namely *European Home Systems Protocol* (*EHS*), *BatiBUS* and *European Installation Bus* (*EIB*). It supports the use of several physical communication media such as *twisted pair* wiring, *power line* networking, *Ethernet*, *infrared* or *radio*. There is a wide range of devices that can be used for controlling a *KNX* network.

- **LonWorks:** Developed by *Echelon Corporation* [117] in 1990, *LonWorks* (*Local Operating Network*) [118] is a widely used fieldbus standard; since 2008, the technology is standardised as *ISO/IEC 14908* [119]. *LonWorks* supports *twisted pair* cabling and *power line* networking and can be controlled by any general-purpose processor which can use the *LonTalk* protocol that is employed by *LonWorks*.

- **BACnet:** *BACnet* (*building automation and control networks*) [120] is another communications protocol which is internationally standardised in *ISO 16484-5* [121]. It employs several communication means such as *Ethernet*, *BACnet/IP*, *Point-to-point connections* over *RS-232* or *LonWorks*' *LonTalk*.

- **LCN:** *LCN* (*Local Control Network*) [122] is a proprietary home automation system developed by *Issendorf KG* in 1992. *LCN* is organised into LCN *modules* that exchange data over a single data wire and the neutral wire power supply.

3.2.2 *KNX* sensors

Each of the *fieldbus* systems mentioned in section 3.2.1 provides a variety of sensors. *KNX*, which is an open standard, is used in this section as an example to enumerate the weather sensors that are available for *fieldbus* systems.

KNX weather sensors are available for

- atmospheric pressure (*barometer*),
- brightness (*photometer*),
- humidity (*hygrometer*),
- precipitation (*rain gauge*),
- solar radiation (*pyranometer*),
- temperature (*thermometer*), and
- wind direction and wind speed (*wind vane* and *anemometer*, respectively).

74-75 2 notes:

There are *weather stations* that provide sensors for a combinations of the above weather elements.

76 07.08.13 by: Mario J..

Other sensors that are available under the *KNX* standard include sensors for water, fluid level, smoke or CO_2 concentration. However, these are not relevant in the context of *SmartHomeWeather* as they do not provide weather data.

3.3 Service data

77 07.08.13 by: Mario J..

This section digs into the details of a number of popular weather services that are available over the Internet. In a first step, a number of aspects are identified that are relevant for the use of a weather service in the context of smart homes. The weather services are then evaluated regarding these aspects.

3.3.1 Available Internet services

78 07.08.13 by: Mario J..

There is a tremendous amount of services providing weather data over the Internet[1]. However, only a small number of these services is suitable for use in the context of *SmartHomeWeather*. The services differ regarding the way data is provided, the data format, the area being covered, the terms of use etc.

The access to Internet weather services in *SmartHomeWeather* is implemented in Java, hence an important question about a certain weather service is whether in can easily be accessed from within a Java program.

For evaluation of Internet weather services, the following aspects are examined:

- **Coverage area:** Which part of the world is covered by the weather service?
  In order to keep things simple, a service covering a larger part of the world is preferred over a service covering a smaller part.

[1] No website has been found that compiles a comprehensive list of Internet weather services; however, there are many websites that discuss various services [123–125]

- **Data format:** In what format is weather data being delivered? Can it be easily parsed and processed?
  A data format that is easier to handle on client-side is preferred over a data format that requires more complicated handling. While *XML* [126] can be handled natively within many programming languages including Java (using *DOM* [127] or *SAX* [128], *JSON* [129] may require an additional library in some languages such as Java[2], while *PHP* comes with built-in support for *JSON* [133]. There may even be data formats that require the development of a new parser.

79 07.08.13 by: Mario J..

- **Data access:** How does the access to weather data work? How is data requested and how is the answer being received?
  The less complicated a request is, the better the service is suitable for *SmartHomeWeather*. Requests in *HTTP* [134] are preferred due to the simplicity of performing requests on the client side in Java using the *Apache HttpComponents project* [135].

80 07.08.13 by: Mario J..

- **Access restrictions, terms of use:** Is the service available freely or is the access restricted? Are credentials (e.g. a username and a password or an access key) required for access? If yes, can the credentials be obtained in a simple way or does that entail a complicated procedure? Are there any access fees for academic or commercial use?
  A service being less restricted is preferred over a service coming with more restrictions.

- **Documentation:** Is there any documentation for this service? Does it cover all aspects or are there some features that are undocumented?
  Of course, a service without documentation is unsuitable. A better documented service is always preferred over a service that is less documented.

- **Stability:** Can one expect the service to remain unchanged over a reasonable amount of time (e.g. several years)? If there will be future changes, will they be announced? How long will they be announced in advance?
  A stable service is preferred over an unstable one. A better handling of changes is preferred over a worse handling of changes.

- **Weather elements:** Which weather elements (e.g. temperature, relative humidity, dew point etc.) are covered by the service?
  A weather service covering a wider range of weather elements is preferred over a service covering a smaller range.

- **Time frame:** Are forecasts available? If yes, how detailed are these forecasts? How far into the future are forecasts available? What is the interval between two forecasts?
  Forecasts for at least 48 hours are mandatory. Over the next 48 hours, there should be at least six forecasts with an interval of at most eight hours between two consecutive forecasts. A weather service covering a longer period than 48 hours into the future and/or more than six forecasts within the next 48 hours is preferred.

81 07.08.13 by: Mario J..

[2] *JSON* libraries for Java include *JSON-lib* [130], *FlexJSON* [131] and *Gson* [132]

- **Weather updates:** How often is the weather data being updated?
  The data should be updated at least every six hours. A service having a shorter update interval is preferred over a service having a longer interval.

82-83
2 notes:

In the following sections, some of the most popular Internet weather services are evaluated. There may be other services that are popular as well while some of the services covered here may actually be not popular at all. The selection is just a set of services that seemed to be popular at the time the evaluation was performed.

In addition to the aspects listed above, some general information is provided (operator and web page). The following sections aim at determining which Internet weather service best fits the given requirements. Furthermore, based on which weather elements are provided by various services, a set of weather elements is determined that will be used in the ontology. Including weather elements that are not available from any weather services are useless.

84
07.08.13
by: Mario J..

Weather services being evaluated Table 3.1 lists all weather services that have been evaluated together with their operators, web pages and coverage areas. *Google Weather API* is discontinued [136]; it is included here because *Google* announced its shutdown after the time this evaluation was conducted. The *Google Weather API* serves as an example of a weather service that is unsuitable for usage in *SmartHomeWeather* as it is discontinued. *DWD* [137] and *Weather Underground* [138] are the only services that do not provide worldwide data.

85
07.08.13
by: Mario J..

Table 3.2 lists all weather services together with the data format they use to provide their data. For the case of Java, *XML*-based formats including *RSS* feeds [139] are preferred because of the built-in support for *XML* by *DOM* and *SAX*. There are libraries for providing *JSON* and *JSONP* [140] support for Java. *CSV* (*comma-separated values*) [141] is a format that can be parsed easily in any language, although using *CSV* leads to more implementation costs than *XML*. There are no libraries available that provide Java support for the *SYNOP* format (*Surface Synoptic Observations* [142] which is used by *DWD*; hence this format is unsuitable to be used in a Java application.

METAR is a format for reporting weather data [143] that is standardised by the *International Civil Aviation Organisation* (*ICAO*) [144]. It is primarily used in aviation to provide weather data to pilots of aircraft. There are a few parsers available, e.g. *PyMETAR* [145] for Python or an implementation for Java [146].

Regarding the terms of use that are summarized in table 3.3, there are some services that require the creation of an account (*DWD*, *World Weather Online*, *Weather Underground*, and *Weather.com*) while others don't (*Yahoo! Weather*, *Google Weather Feed*, *yr.no*, and *METAR*). Some services provide all of their data freely (*DWD*, *yr.no*, *METAR*, or *NWS*), others provide the data freely only for non-commercial purposes (*Yahoo! Weather*, *World Weather Online*, and *Weather Underground*); for some services, the terms of use are unknown (*Google Weather Feed*, *Weather.com*).

86
07.08.13
by: Mario J..

Except the ones not providing information about changes (*Yahoo! Weather*, *World Weather Online*, *Weather Underground* and *Weather.com*) and the one being discontinued (*Google Weather Feed*), all weather services declare to perform regular updates to their weather services (see table 3.4). Any changes are announced either on the web pages, via *RSS* feeds or via email.

Name	Operator	Web page	Coverage area
DWD	Deutscher Wetterdienst (*DWD*, "German weather service")	[137]	Worldwide (current weather data); Germany and large cities around the world (forecasts)
Yahoo! Weather	Yahoo! Inc.	[147]	Worldwide
World Weather Online	World Weather Online	[148]	Worldwide
Google Weather Feed	Google Inc.	none[1]	Worldwide
yr.no	Meteorologisk institutt (Norwegian Meteorological Institute), Norsk rikskringkasting AS (NRK, Norwegian Broadcasting Corporation)	[149]	Worldwide
METAR	Airports around the world	[150]	Worldwide
NWS	National Weather Service[2]	[151]	Only US
Weather Underground	Weather Underground	[138]	Worldwide
Weather.com	The Weather Channel, LLC	[152]	Worldwide

[1] not publicly advertised *API*
[2] part of *NOAA* (*National Oceanic and Atmospheric Administration*) [153]

Table 3.1: Names, operators, web pages, and coverage areas of all weather services that have been evaluated.

Table 3.5 shows which data is available from the weather services. While temperature is available from all services, some services provide only data about a few other weather elements (*Google Weather Feed* and *Weather Underground*); from some services, only limited forecasts are available (*DWD*, *Yahoo! Weather*, and *World Weather Online*, *Weather.com*). Some services limit the availability of data via their freely accessible interface (*World Weather Online* and *Weather Underground*).

As seen in table 3.6, *METAR* only provide current weather data; some services provide forecasts for several days, but give only one forecast per day (*Yahoo! Weather*, *Google Weather Feed*, and *Weather Underground*). All weather services provide frequent updates (at least every few hours).

Nearly all weather services discussed provide comprehensive documentation regarding their interfaces and use; the only exception is the *Google Weather Feed* which was an inofficial *API* and therefore has never had any official documentation.

Name	**Data format**	**Data access**
DWD	*SYNOP* (current weather data); not machine-readable weather maps, tables and texts (forecasts)	*FTP* [154]
Yahoo! Weather	*RSS* feed	*HTTP*
World Weather Online	*XML*, *JSON*, *JSONP*, *CSV*	*HTTP*
Google Weather Feed	*XML*	*HTTP*
yr.no	*XML*	*HTTP*
METAR	Format standardised by *ICAO*; not human-readable.	Via HTML from *NOAA*'s web page (*National Oceanic and Atmospheric Administration*). Various other data sources are available.
NWS	*XML*	*SOAP* [155], *REST* [156]
Weather Underground	*JSON*, *XML*	*HTTP*
Weather.com	*JSON*, *XML*	*HTTP*

Table 3.2: Data formats and protocols for data access of weather services.

90
07.08.13
by: Mario J..

3.4 Conclusion

Table 3.7 summarizes the main advantages and disadvantages of the weather services discussed in 3.3.1 regarding the requirements listed there. Based on that evaluation, it is determined that *yr.no* clearly suits the requirements of smart homes and *SmartHomeWeather* best. It provides current data and four forecasts per day over a period of nine days. Data is accessible in *XML* format via *HTTP* and is licensed under *CC-BY 3.0* [157]. Data includes details about temperature, wind direction and speed, chance and intensity of precipitation, atmospheric pressure, relative humidity, cloud coverage, and fog.

Thus, *yr.no* is used in chapter 6 for developing the reference implementation of a program that obtains weather data for a certain location and feeds it into the *SmartHomeWeather* ontology. See section 3.5 for details on how to query *yr.no*'s weather *API*.

Nevertheless, the source of weather data shall remain replaceable. The ontology is designed in a way that makes switching to another weather service simple. If *yr.no* is to be replaced by another weather service, the ontology remains unchanged; only the program that imports weather data must be adapted. If the weather service used does not support one or more weather elements, they are simply omitted in the program's output.

91-92
2 notes:

93
07.08.13
by: Mario J..

The ontology uses these weather elements which are available from many Internet weather services and many weather sensors (in no particular order):

Name	**Access restrictions, terms of use**
DWD	An account is mandatory for access. Account creation is a matter of minutes and new accounts do not need to be approved by DWD. The use of DWD is free for use within smart home projects.
Yahoo! Weather	Free for use for individuals and non-profit organizations, attribution required. No statement about commercial use anywhere in the documentation.
World Weather Online	An account must be created for access to both the Free API and the Premium API. The Free API is free to use for personal and commercial use, credits must be given to World Weather Online. Costs for the Premium API start at 20 US dollars per month.
Google Weather Feed	No account required; terms of use unknown (does not have an explicitely stated licence due to being an inofficial interface)
yr.no	No account required; data licenced under *CC-BY 3.0* [157]
METAR	Freely available without restrictions
NWS	Freely available without restrictions
Weather Underground	Account creation is mandatory; available for personal, non-commercial use (public API); custom weather services are available
Weather.com	An account must be created in a complicated process; licence unclear

Table 3.3: Access restrictions and terms of use for weather services.

- Temperature,
- relative humidity,
- dew point,
- cloud cover,
- chance and intensity of precipitation,
- speed and direction of wind,
- atmospheric pressure, and
- solar radiation.

Furthermore, the overall weather condition is added to the ontology. This weather condition is described by the following states: *Cloudy*, *fog*, *light clouds*, *partly cloudy*, *rain*, *sleet*, *sun*, *thunder*. At most three of them may be used together to describe the overall weather condition; two contradictory states, e.g. *light clouds* and *partly cloudy*, may not be used at the same time. These states are taken from the list of symbols used for describing the weather condition in the *API* of *yr.no* [158].

Name	**Stability**
DWD	A few changes every weeks which are announced via email at least one week in advance; nothing about the data required by *SmartHomeWeather* has been changed during the last twelve months.
Yahoo! Weather	Unknown
World Weather Online	Unknown
Google Weather Feed	Discontinued [136]
yr.no	New releases of the API one or two times per year; the old API remains in operation a few months after the release of a new one. Announcements are made on the web page. A less frequently changing API with *long term support* is available.
METAR	Stable.[1]
NWS	Small changes every few months with announcements via *RSS*. Most of the interface remains unchanged in the long run.
Weather Underground	The has been a recent API change; the stability is unknown.
Weather.com	Unknown

99-100
2 notes:

[1] *METAR* is standardized by *ICAO* [144]. It was introduced in 1968 and has been modified a numer of times since. However, most elements remained the same since introduction and can be expected to remain unchanged in the long run.

101
07.08.13
by: Mario J..

Table 3.4: Stability of weather services.

102
07.08.13
by: Mario J..

Additionally, the ontology supports the position of the sun. This data is not obtained from any sensor or service; a few algorithms are available for calculating the sun's position given date, time and the geographical position (latitude and longitude); see section 3.6 for details.

In case a weather service does not provide a value for the dew point, there are simple means available for calculating that value from the values of temperature and relative humidity. E.g. the following formula leads to a dew point value with an accuracy sufficient for smart homes [159]:

$$t_d \approx t - \left(\frac{100 - RH}{5} \right)$$

t_d represents the dew point temperature, t the air temperature and RH the relative humidity in percent (i.e. in the interval $[0, 100]$). Accordingly, the value of relative humidity can be calculated from the values of dew point and temperature using the same method (the case of a missing temperature values, but values of dew point and relative humidity is unlikely).

103
07.08.13
by: Mario J..

As the atmospheric pressure decreases with increasing altitude above sea level, it is necessary to convert the pressure value observed by a sensor to the equivalent pressure at sea level

Name	Weather elements
DWD	Temperature, relative humidity, wind speed and direction, precipitation, atmospheric pressure, cloud coverage (current weather). Temperature and weather condition (forecasts).
Yahoo! Weather	Temperature, wind chill, speed and direction, relative humidity, visibility, atmospheric pressure, sunrise, and sunset times (current weather). Temperature and weather condition (forecast).
World Weather Online	Temperature, wind speed and direction, relative humidity, and precipitation intensity (Free API). Additionally dew point, chance of precipitation and atmospheric pressure (forecast).
Google Weather Feed	Temperature, humidity (only current weather), weather condition.
yr.no	Temperature, wind direction and speed, chance and intensity of precipitation, atmospheric pressure, relative humidity, cloud coverage, and fog.
METAR	Temperature, dew point, wind speed and direction, precipitation, cloud coverage, visibility, and atmospheric pressure.
NWS	Temperature, dew point, wind speed and direction, precipitation, and cloud coverage.
Weather Underground	Minimum and maximum temperature, weather condition (public API); more data via custom weather services
Weather.com	Temperature, humidity, precipitation, wind speed and direction, sunrise and sunset times, and the weather condition (current weather and forecasts); atmospheric pressure, visibility, and dew point (current weather only)

Table 3.5: Weather elements provided by weather services.

Name	Time frame	Weather updates
DWD	72 hours	Every few hours
Yahoo! Weather	48 hours (only one forecast per day)	Every few hours
World Weather Online	Five days	Every 3 to 4 hours
Google Weather Feed	72 hours (only one forecast per day)	Every few hours
yr.no	Nine days (four forecasts per day)	Every few hours
METAR	Only current weather data	Every 30 minutes
NWS	More than seven days	About once per hour
Weather Underground	Five days (one forecast per day)	Every few hours
Weather.com	Five days	Every few hours

Table 3.6: Time frame of forecast data and frequency of weather updates at weather services.

Weather service	**Advantage(s)**	**Disadvantage(s)**
DWD	-	Data format (*SYNOP*); forecasts lack important weather elements
Yahoo! Weather	Simple data format (*RSS*)	Low number of forecasts; forecasts lack important weather elements
World Weather Online	Wide range of data formats	Free *API* lacks important data
Google Weather Feed	Simple data format (*XML*)	Unofficial, undocumented *API*; unknown terms of use; only a small number of available weather elements; discontinued
yr.no	Simple data format (*XML*); available under *CC-BY 3.0* [157]	-
METAR	-	Data format (*METAR*); no forecasts
National Weather Service	Simple data format (*XML*)	No worldwide coverage
Weather Underground	Data formats (*XML* and *JSON*)	Public *API* lacks important data
Weather.com	Data formats (*XML* and *JSON*)	Complicated account creation

Table 3.7: Advantages and disadvantages of Internet weather services

104 07.08.13 by: Mario J..

using the *Barometric formula*. is not necessary for many Internet weather services (including *yr.no*) as these often report the value that has already been converted.

105 07.08.13 by: Mario J..

To ensure the ontology can be used in the desired manner, it is necessary that the competency questions from section 3.1 can be answered by the ontology using the provided input data. See table 3.4 for details about which competency questions can be answered by using which weather element(s). Section 5.7 will discuss whether *SmartHomeWeather* can actually answer the competency questions.

3.5 Weather data *API* of *yr.no*

This section gives an overview on how requests to the weather *API* of *yr.no* work.

106 07.08.13 by: Mario J..

For an arbitrary request, latitude and longitude must be specified (in degrees; northern latitudes and eastern longitudes are represented by positive values) [160]. Additionally, the altitude above sea level (in metres) may be specified for locations outside Norway. Based on this input data, a *URL* of the format

Competency question	**Weather element(s)**
What is the current weather situation?	*all available elements*
What will the weather situation be in one hour, in two hours, ..., in 24 hours?	*all available elements*
What is the current temperature, humidity, wind speed, ...?	*the corresponding weather element*
What will be the temperature, humidity, wind speed, ... in one hour, in two hours, ..., in 24 hours?	*the corresponding weather element*
What will be the minimum temperature, humidity, ... over the next 24 hours? What about maximum values?	*the corresponding weather element*
Will the weather change? Will the temperature, humidity, ... rise or fall?	*the corresponding weather element*
Does it rain? Will it rain in the next hours? Will it rain today?	Precipitation
Will there be sunshine today?	Cloud coverage, sun radiation, sun position
Do we need to irrigate the garden?	Precipitation, cloud coverage, sun radiation
Will there be severe weather?	Wind, precipitation, temperature
Will temperature drop/stay below 0 °C?	Temperature
When can we open windows and when do we have to keep them shut?	Precipitation, wind
When do we need sun protection?	Sun radiation, cloud coverage, sun position
When will it outside be colder than inside the house? When will it be warmer?	Temperature

Table 3.8: Assignment of competency questions to weather element(s)

107
07.08.13
by: Mario J..

```
http://api.yr.no/weatherapi/locationforecast/1.8/?lat=<latitude>;lon=<longtude>
```

108-109
2 notes:

or

```
http://api.yr.no/weatherapi/locationforecast/1.8/?lat=<latitude>;lon=<longtude>;msl=<altitude>
```

is constructed, e.g.

```
http://api.yr.no/weatherapi/locationforecast/1.8/?lat=48.21;lon=16.37;msl=171
```

for the city of Vienna, Austria (N 48.21°, E 16.37°, 171 m above *MSL*). An *HTTP GET* request to this *URL* returns an *XML* document conforming to the *XML Schema* [44] definition that can be found online [161].

The structure of this *XML* document is as follows (attributes are omitted for better readability):

```
<weatherdata>
```

```
        <meta>
                <model />
        </meta>
        <product>
                /* ... */
        </product>
</weatherdata>
```

The attributes of the `<model>` element describe when the forecast has been created, when it will be updated for the next time and what date and time of the first and the last forecast returned are.

There is an arbitrary number of `<time>` elements that are children of the `<product>` element. Every `<time>` element represents the weather forecast for a certain period of time. Each `<time>` element has a `<location>` element that has a child element for each weather property.

This is a typical `<time>` element:

```
<time datatype="forecast" from="2013-06-24T12:00:00Z" to="2013-06-24T12:00:00Z">
        <location altitude="171" latitude="48.2100" longitude="16.3700">
                <temperature id="TTT" unit="celcius" value="15.7"/>
                <windDirection id="dd" deg="303.4" name="NW"/>
                <windSpeed id="ff" mps="6.7" beaufort="4" name="Laber bris"/>
                <humidity value="67.5" unit="percent"/>
                <pressure id="pr" unit="hPa" value="1016.0"/>
                <cloudiness id="NN" percent="100.0"/>
                <fog id="FOG" percent="0.0"/>
                <lowClouds id="LOW" percent="100.0"/>
                <mediumClouds id="MEDIUM" percent="89.1"/>
                <highClouds id="HIGH" percent="43.0"/>
        </location>
</time>
```

In total, there are 41 elements that are allowed to be children of the `<location>` element. The ones that are relevant for *SmartHomeWeather* are: `pressure`, `precipitation`, `cloudiness`, `lowClouds`, `mediumClouds`, `highClouds`, `temperature`, `dewpointTemperature`, `humidity`, `windDirection`, `windSpeed`, and `symbol`. While other elements may occur in the weather reports for some locations (especially for locations within Norway), these are the elements which are used worldwide.

110
08.07.13
by: Mario J..

None of the child elements is required. However, for most places of the world, the *XML* document contains `<location>` elements having two different sets of child elements:

- Some `<location>` elements have the child elements `temperature`, `windDirection`, `windSpeed`, `humidity`, `pressure`, `cloudiness`, `fog`, `lowClouds`, `mediumClouds` and `highClouds` (as shown above). The values of the attributes from and to of the enclosing `<time>` element are equal.

111
07.08.13
by: Mario J..

- Some `<location>` elements have the child elements `precipitation` and `symbol`; the values of the attributes `from` and `to` of the enclosing `<time>` element differ by three to six hours; such an element looks like this:

```
<time datatype="forecast" from="2013-06-24T06:00:00Z" to="2013-06-24T12:00:00Z">
        <location altitude="171" latitude="48.2100" longitude="16.3700">
                <precipitation unit="mm" value="2.5"/>
                <symbol id="LIGHTRAIN" number="9"/>
        </location>
</time>
```

The content of the *XML* document covers a period of nine days, starting at the current day.

3.6 Position of the sun

Besides location-based weather data, *yr.no* offers an interface for retrieving sunrise and sunset data [162]. Given a position in latitude and longitude together with a date, the times of rise and set of sun and moon are provided. The angle of the sun at solar noon is also given.

112 07.08.13 by: Mario J..

However, these data are inappropriate for use within the *SmartHomeWeather* ontology. For data about the sun's position to be of any value, it is necessary that the position is known for every *Weather state*. Furthermore, most of the weather services that are available via Internet do not offer any data about the sun's position.

There are several algorithms for calculating the sun's position (specified by zenith and azimuth angles) at a certain location given by latitude and longitude at a certain time; two of them are:

- The *SPA algorithm* [163] provides results for zenith and azimuth angles with uncertainties of less than 0.0003 degrees in the period from 2000 BC to 6000 AD. However, the algorithm which is complicated to implement. Hence, it is used only in contexts that require values of that precision.

- A algorithm that can be implemented more easily is the *PSA algorithm* [164]. Its results differ from the actual values more than the results calculated by the *SPA algorithm*: For the period between 1999 and 2015, the differences per value are smaller than 0.01 degrees. As this level of accuracy is sufficient for use in a smart home, the *PSA algorithm* is suitable for use by the *Weather importer* application (see section 6.2.1). A ready-to-use implementation of the *PSA algorithm* in C++ is available; this implementation can easily be ported to Java.

113 07.08.13 by: Mario J..

CHAPTER 4

Methodologies for developing ontologies

114
07.08.13
by: Mario J..

There are numerous methodologies for building ontologies. As this chapter points out, all of them strive for avoiding common pitfalls [165] and try to minimise the need for refactorisation at late development steps. Each approach forces the ontology designer to determine as many details about the ontology's domain as early as possible.

All methodologies have in common that knowledge acquisition is centered around *competency questions* that roughly define a scope for the ontology and details about that scope [166]. Competency questions are stated at the very beginning of the design process and provide the basis for all further steps towards the ontology. The ontology can be considered complete if it is able to provide answers to all competency questions (except the ones that cannot be answered by an ontology).

In the article about their approach towards ontology design, Noy and McGuinness state three fundamental rules [105] of ontology design. Although the authors only apply them to their own approach, they hand out advice for many design decisions, regardless of which approach is used for the design of the ontology:

> *1) There is no one correct way to model a domain – there are always viable alternatives. The best solution almost always depends on the application that you have in mind and the extensions that you anticipate.*
>
> *2) Ontology development is necessarily an iterative process.*
>
> *3) Concepts in the ontology should be close to objects (physical or logical) and relationships in your domain of interest. These are most likely to be nouns (objects) or verbs (relationships) in sentences that describe your domain.* [105]

This chapter discusses some existing methodologies for the construction of ontologies. Among the methodologies that can be found in literature, the ones discussed as candidates for the design of *SmartHomeWeather* are

- the one by Uschuld and King [106] (section 4.2.1),
- the method used by Grüninger and Fox for the *TOVE* (*TOronto Visual Enterprise*) ontologies [107] (section 4.2.2),
- *Ontology 101* by Noy and McGuinness [105] (section 4.2.3),
- the *UPON* (*Unified Process for ONtology building*) methodology by De Nicola, Missikoff and Navigli [167] (section 4.2.4), and
- *METHONTOLOGY* by Gómez-Pérez et al. [108] (section 4.2.5).

There are several other methodologies that are not covered here, such as *Model Driven Ontology* [168], the *NeOn Methodology* [169], the approach of Berneras et al. in the context of the *Esprit KACTUS project* [170], the methodology based on the *SENSUS ontology* [171], and a method [172] based on *Formal Concept Analysis* [173].

4.1 Evaluating ontology development methodologies

The evaluation in this chapter is loosely based on the article by Fernández-López that evaluates a set of methodologies for building ontologies [174]. There are several other articles that cover this topic [175–177].

For each methodology, the following topics are discussed in section 4.2:

- **Description:** Each step of the methodology is presented.
- **Applications:** Some ontologies that have been developed using the methodology are enumerated, if any. Applications may include both cases where the methodology was just applied to provide detailed insights into the methodology itself and cases where the methodology was used for the development of an ontology as part of a project.
- **Analysis:** The methodology is analysed regarding a pre-defined set of criteria (see below):
  - **Effort:** Different approaches lead to different efforts for the development of ontologies. Although the minimisation of the effort is not a target of *SmartHomeWeather*, it is unnecessary to apply an approach which leads to an enormous development effort compared to other methodologies.
  - **Usage:** If an approach is widely used, this may indicate that the approach is considered suitable for ontology development by many designers. The other way, a seldomly applied methodology may be inappropriate for most use cases.
  - **Applicability:** An approach may be limited to certain kinds of domains and be unsuitable for designing *SmartHomeWeather*.
  - **Strictness:** Due to the fact that there is no one correct way to design an ontology, every approach must leave a certain margin to the ontology designer to decide about implementation details. However, a margin being too wide may lead to an inaccurate or incomplete ontology.

- **Formality:** The ontology design process can reside on an *informal* level (the ontology and all artefacts created during development are described using natural language, in tables, and in diagrams), a *formal* level (all aspects of the ontology are described using the logical model of the ontology language the ontology is intended to be implemented in), or anything in between.
- **Level of detail:** The level of detail of the description of the design process can range from giving just an overview to a level describing every step in a very detailed manner.
- **Documentation:** Methodology may enforce the creation of documentation while others delegate the decisions about how the documentation is structured and what is documented to the ontology designer. This may lead to missing, inaccurate, or incomplete documentation.

Section 4.3 then compares the methodologies and concludes about the results and comes to a decision regarding the methodology which fits the requirements of *SmartHomeWeather* but and thus is used for the development of the ontology. It is possible that this decision is not be unambiguous if more than one approach turns out to be suitable for the present context.

As section 4.2 focuses on the characteristics of the methodologies required to take a decision in favour of one of the approaches, section 4.4.2 then describes the selected approach in a more detailed manner.

4.2 The ontology development approaches

4.2.1 Methodology by Uschold and King

Description

When Uschold and King published their approach in 1995 [106], it was among the first methodologies proposed towards the development of new ontologies.

This approach divides ontology development into a set of stages, as depicted in figure 4.1:

- *Identify Purpose*: At the very beginning, the purpose of the ontology needs to be identified: Why is the ontology built, what are its intended uses and what is its scope? A set of *competency questions* are formulated.

- *Building Ontology*: This stage is divided into three sub-stages:
  - *Ontology Capture*: Concepts and relationships are identified and textually defined. Furthermore, terms which refer to these concepts and relationships are defined.
  - *Ontology Coding*: In this step, the representation of the ontology from the *Ontology Capture* stage is transformed into a formal (ontology) language (e.g. *OWL*), presumably using an ontology development environment (such as *Protégé*).
  - *Integrating Existing Ontologies*: As the ontology being developed partly covers the scope of ontologies that are already exist, research has to be done which ontologies can be reused.

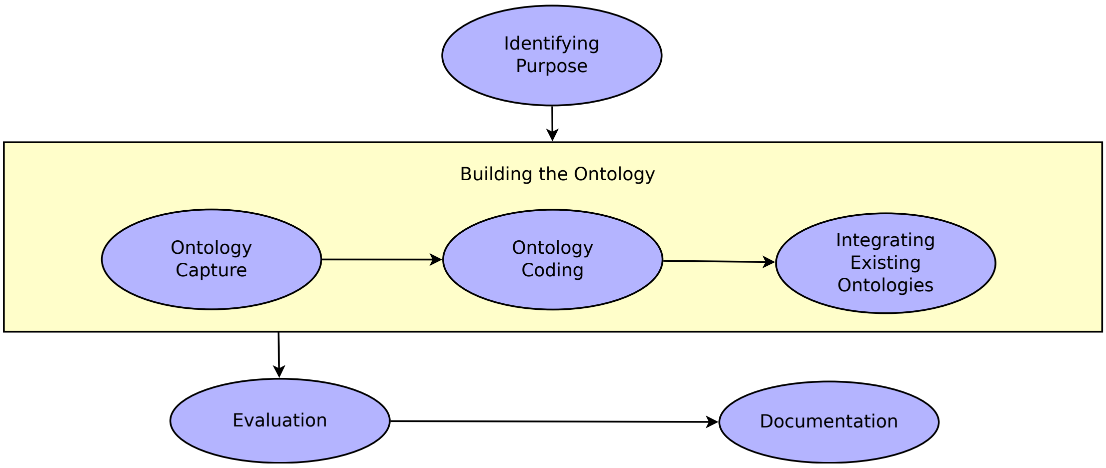

Figure 4.1: The workflow proposed by the methodology by Uschold and King [106].

- *Evaluation*: During this stage, it is verified whether the ontology has the ability to fulfil the purpose that was initially identified.

- *Documentation*: Finally, all results from the previous stages must be documented thoroughly. This approach does not enforce documentation throughout the development process; this may lead to incomplete, inaccurate, or missing documentation.

As this approach is one of the first comprehensive methodologies for ontology development, it lacks the experience that has been carried together by ontology designers over the recent years. The descriptions of the proposed steps hardly dig into details. However, its overall structure matches most of the later approaches; furthermore, it is an easily comprehensible approach.

Applications

There are various applications of this methodology. Some examples include the *Enterprise Ontology* which models the domain of business enterprises [178] (probably the most prominent example based on this methodology), the *e-Business Model Ontology* that covers processes found in businesses available over the Web [179], and the LKIF *Core Ontology of Basic Legal Concepts* (*Legal Knowledge Interchange Format*) [180].

Analysis

- **Effort:** The effort caused by using this approach can be considered to be low, compared to other approaches such as *METHONTOLOGY* (see section 4.2.5) or the *UPON* methodology (refer to section 4.2.4).

- **Usage:** There are several applications of this approach which qualify the approach for being suitable for many different knowledge domains.

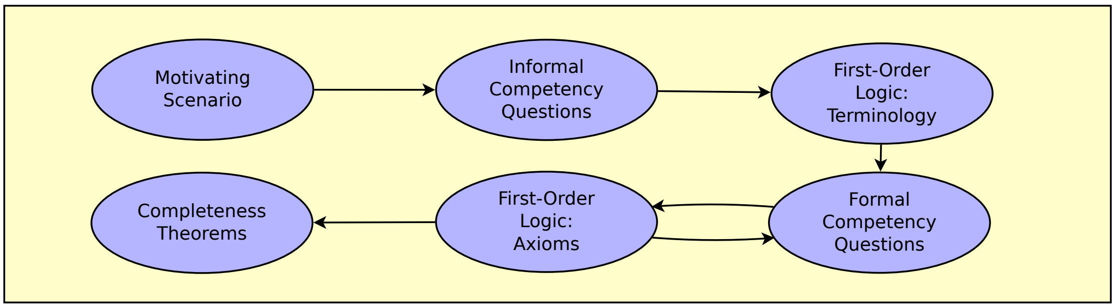

Figure 4.2: The workflow proposed by the *TOVE* methodology [107].

- **Applicability:** There are no restrictions regarding the application of this approach to various domain.

- **Strictness:** The approach does not describe a strict process; there are wide margins for individual decisions by the ontology designer. However, this renders decisions possible that may affect the functionality of the ontology in a negative way.

- **Formality:** This is an informal approach involving natural language descriptions.

122 07.08.13 by: Mario J..

- **Level of detail:** The description of this approach does give an overview, but does not dig into the details of each step.

123 07.08.13 by: Mario J..

- **Documentation:** The process proposed by this approach includes a *Documentation* step; however, it does not enforce the creation of documentation artefacts throughout the development process. Additionally, no details about how structure the documentation are defined.

4.2.2 Methodology by Grüninger and Fox (*TOVE*)

Description

When Grüninger and Fox began designing their *TOVE* (*TOronto Visual Enterprise*) *Enterprise Modelling project* based on ontologies, they failed to identify an already existing approach towards ontology development that would suit their needs. In order to overcome this problem, they formulated their own approach [107].

This approach splits the ontology development process into a set of activities as shown in figure 4.2:

- The *Motivating Scenarios* are a set of use cases the ontology is used for.

- These scenarios lead to *Informal Competency Questions* which are a set of questions that the ontology shall be able to answer once it is completed.

- *First Order Logic: Terminology*: In this step, the terminology is specified using first-order logic (or the logic used by the intended ontology language) [21]. The terms (objects, attributes, and relations) are derived from the previously formulated competency questions.

- Then, the competency questions are transformed into *Formal Competency Questions* in terms of entailment and consistency problems with respect to the axioms in the ontology.

- *First Order Logic: Axioms*: Now, formal axioms are stated that define the terms and constraints on objects in the ontology using first-order logic.

- Finally, *Completeness Theorems* are defined w[illegible]h are used to prove whether the ontology is complete (i.e. it can provide answers to all

124
07.08.13
by: Mario J..

Once all these steps are completed, the formal model is implemented using a ontology language (e.g. *OWL*).

Applications

Examples for the application of this methodology are the *ontology of virtual humans* [181], an *ontology for software maintenance* [182], and an ontology that forms the base of an expert system for corporate financial rating [183].

Analysis

The approach proposed by Grüninger and Fox is a rather formal one that is based on first-order logic which forms the base of many ontology languages.

- **Effort:** Putting all parts of the ontology bei[illegible]veloped into a formal model in first-order logic may turn out to be a tedious task. The

125
07.08.13
by: Mario J..

- **Usage:** Several applications of this approach can be found in literature. However, there are approaches such as *METHONTOLOGY* (see section 4.2.5) or the approach by Uschold and King (see section 4.2.1) that are far more popular than this approach.

- **Applicability:** There are no limits regarding the domains this approach can be used with.

- **Strictness:** This approach defines a strict procedure to follow, but leaves a margin for individual design decisions by the ontology developer.

- **Formality:** This is a rather formal approach that involves definitions based on the logic used by the ontology language that is intended to be used.

- **Level of detail:** The description of this approach gives details about each step, but leaves a margin for the ontology developer who follows the approach.

- **Documentation:** This approach does not enforce the creation of documentation; hence problems regarding missing, incomplete, or inaccurate documentation may arise when applying this approach.

4.2.3 Ontology 101

Description

In their paper *Ontology Development 101: A Guide to Creating Your First Ontology* [105], Noy and McGuiness present an informal and rather intuitive approach for building ontologies from scratch. It is geared towards people without or with little prior knowledge about how to design an ontology and qualifies for demonstrating the essence of ontologies.

The approach is divided into a set of steps:

1. The domain and scope of the ontology are determined. The preferred way for this is to formulate *competency questions* the ontology should be able to answer.

2. Existing ontologies are considered to be reused to avoid doing work that has already been done and to simplify interoperability with other ontologies.

3. Important terms in the ontology are enumerated, i.e. a *glossary of terms* is built.

4. From the glossary in the previous step, all terms that are classes are identified. They are then related to each other in order to create a *class hierarchy*.

126 07.08.13 by: Mario J..

5. The next step iterates all classes and tries to identify terms from the glossary which are properties of the classes.

6. Then, for the properties the ranges of possible values are specified.

7. Finally, instances from the glossary are selected and added to the ontology.

127 07.08.13 by: Mario J..

These steps are not strictly performed one after the other; instead, an iterative process is proposed which iterates the whole set of steps repeatedly. Figure 4.3 depicts the workflow this process.

128-129 2 notes:

Besides the approach itself, *Ontology 101* provides a huge set of rules of thumb which guide the ontology designer towards a well-conceived ontology, advises her of common pitfalls and help her identify both adequate and improper patterns.

Applications

Applications of *Ontology 101* include a *Human Community Ontology* [184], an *Ontology for Intrusion Detection* [185], and an ontology which is part of *BioPAX*, an effort towards improving knowledge exchange in the research of biological pathways [186].

Analysis

130 07.08.13 by: Mario J..

Ontology 101 provides a simple and coherent approach for building ontologies. However, it comes with a few downsides.

- **Effort:** The effort of applying *Ontology 101* to a certain domain can be considered to be low.

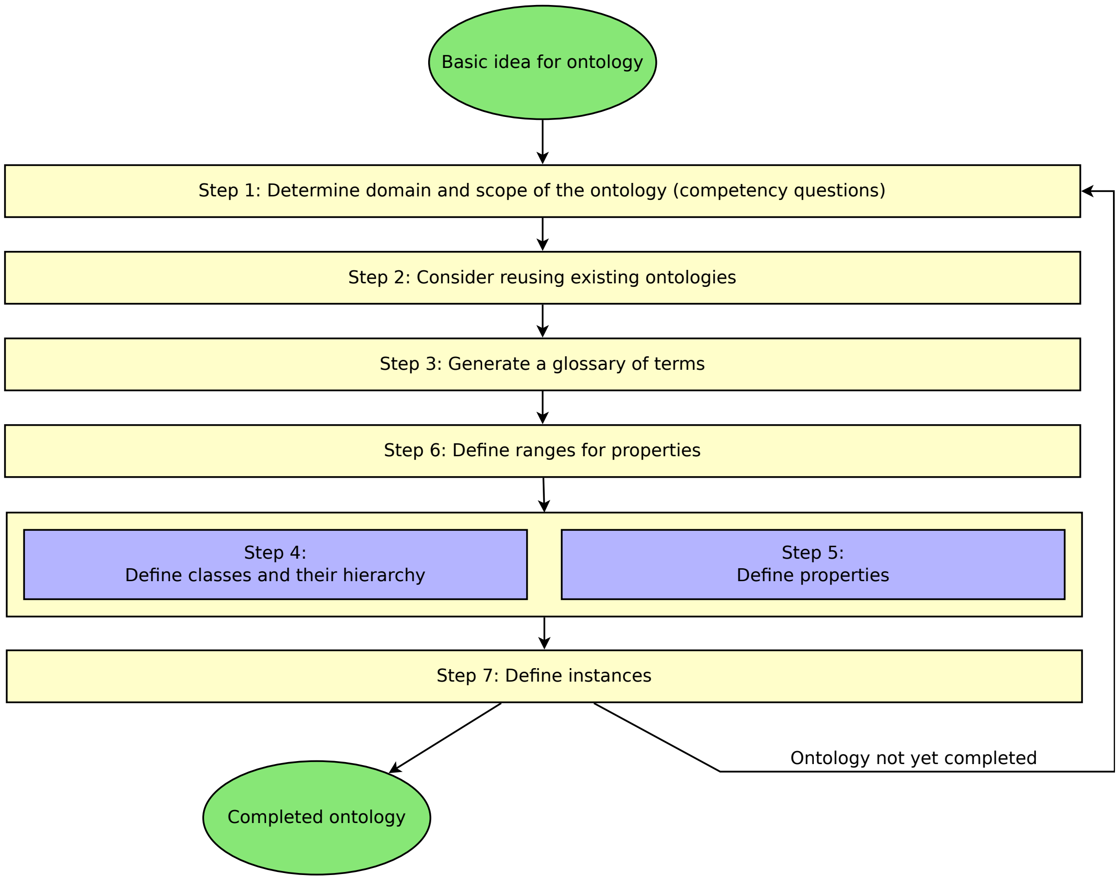

Figure 4.3: The workflow proposed by *Ontology 101* [105].

- **Usage:** The methodology is very often cited in literature to give readers an understanding of the basics of ontology development. However, compared to other methodologies, the number of applications is low.
- **Applicability:** *Ontology 101* does not state any limitations regarding the use of its approach for arbitrary domains.
- **Strictness:** The approach does not enforce strict rules and leaves a broad margin to the ontology designer.
- **Formality:** This is a completely informal approach that does not involve any work with formal models.
- **Level of detail:** The description of this approach gives details about each step, but leaves a margin for the ontology developer who follows the approach.
- **Documentation:** *Ontology 101* completely lacks a description on how documentation is generated; this may lead to missing, incomplete, or inaccurate documentation.

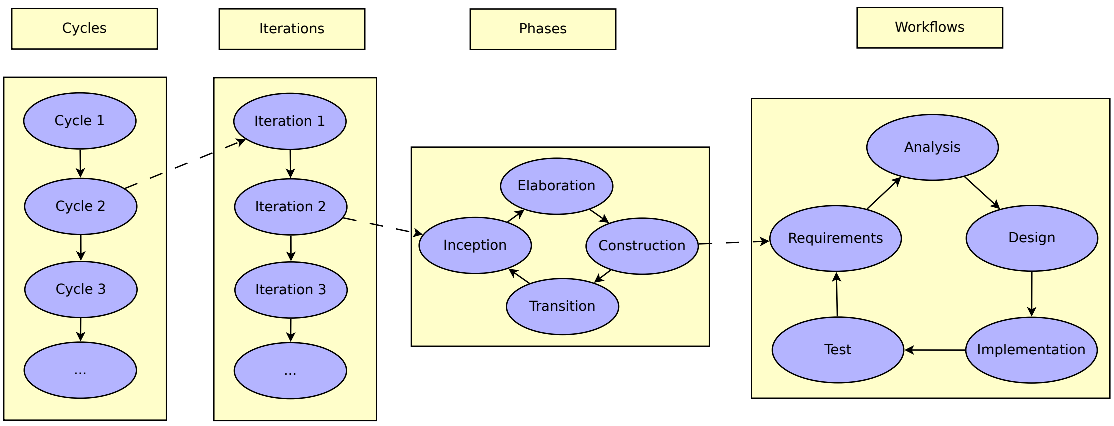

Figure 4.4: The workflow proposed by the *UPON* methodology [167]

4.2.4 The *UPON* methodology

Description

De Nicola, Missikoff, and Navigli observed many similarities between creating software artefacts and the development of ontologies [167]. Therefore, they developed the *UPON* (*Unified Process for ONtology*) methodology which takes advantage of the *Unified Process* [187] and the *Unified Modeling Language* (*UML*) [188], both known from software development.

As the *Unified Process*, *UPON* is *use-case driven*, *iterative*, and *incremental*. As shown in figure 4.4, the *UPON process* consists of *cycles*, *phases*, *iterations*, and *workflows*. The whole process is divided into cycles which each result in a new version of the ontology. Four different phases (*inception*, *elaboration*, *construction*, and *transition*) form each cycle. Each phase in turn is divided into an arbitrary number of iterations; during each iteration, five workflows (*requirements*, *analysis*, *design*, *implementation*, and *test*) take place.

131 07.08.13 by: Mario J..

During the *inception phase*, requirements are captured. The *elaboration phase* comprises the identification and structurisation of fundamental concepts. The *construction phase* involves the creation of elements in the ontology to be created. Eventually, in the *transition phase* testing and evaluation of the work within this cycle takes place.

132 07.08.13 by: Mario J..

133-134 2 notes:

The *requirements workflow* consists of a set of tasks which analyse the knowledge domain, scope and purpose of the ontology, relevant terms, related use cases and competency questions are identified. These results are the input for the *analysis workflow* which refines concepts and their relations and enriches their description with detailed definitions in order to create a *reference glossary*. The goal of the *design workflow* is to model concepts, concept hierarchies, and domain-specific relationships from the previously created *reference glossary* for their later implementation. During the *implementation workflow*, the ontology designed in the previous workflow are translated to an ontology language. Finally, the *test workflow* involves the evaluation of the ontology for *semantic quality* (the absence of contradictory concepts) and *pragmatic quality* (the usefulness of the ontology for the user).

Each of these steps includes the generation of one or more documentation artefacts (e.g.

diagrams or tables) that document the results. Hence, once the *UPON* process is completed, both the ontology and its documentation are completed and match each other. Documentation is not implemented as a separate step to avoid any problems that may arise from such an approach.

Applications

UPON has been applied to create four different ontologies in the context of the *Athena Integrated Project* [189]. Furthermore, applications of *UPON* include an ontology representing word meaning [190], an ontology for mapping individuals and their relationships in a social network [191] and the *LD-CAST reference ontology* which is part of a project that aims at developing a semantic cooperation and interoperability platform for the *European Chambers of Commerce* [192, 193].

Analysis

The *UPON* approach is a deeply structured approach that applies well-established technologies from software development to the field of ontology design. While this approach leads to well-designed ontologies and works well for developing large environments, in the case of *SmartHomeWeather* this is possibly an over-engineered approach.

135-136
2 notes:

- **Effort:** Due to its process which consists of many single steps which generate a huge number of artefacts, the application of the *UPON* methodology leads to a greater effort compared to other approaches such as *METHONTOLOGY* (see section 4.2.5). The smaller the ontology to be developed is, the larger the difference becomes. As *SmartHomeWeather* can be considered to be a rather small ontology, *UPON* may be unsuitable for its design.

137
07.08.13
by: Mario J..

- **Usage:** The paper proposing the *UPON* methodology [167] gets cited often; however, due to its enormous effort, it is harder to find use cases for it compared to other approaches.

- **Applicability:** There are no limitations regarding the application of this approach to various domains.

- **Strictness:** The *UPON* approach is a very strict process, but nevertheless leaves some margin to the ontology designer for individual design decisions.

- **Formality:** This is an informal approach.

- **Level of detail:** The description of this approach is very detailed.

- **Documentation:** Every step during the design process involves the generation of documentation artefacts.

4.2.5 METHONTOLOGY

Description

METHONTOLOGY partitions the development process of an ontology into several activities: *Planification*, *specification*, *knowledge acquisition*, *conceptualisation*, *formalisation*, *integration*, *implementation*, *evaluation*, *documentation*, and *maintenance*.

Because the activity of *planification* is finished by defining the development process of *METHONTOLOGY*, ontology construction starts with *specification* (by creating a *Ontology Requirements Specification Document*) and *knowledge acquisition* about the given domain from all available sources. *Conceptualisation* then generates a set of diagrams and tables that describe
138 different aspects of the ontology. These artefacts are the *glossary of terms*, diagrams showing
07.08.13 the *concept taxonomies*, *binary relation diagrams*, the *concept dictionary*, and tables for *bi-*
by: Mario J.. *nary relation details*, *instance attributes*, *class attributes*, *constants*, *formal axioms*, *rules*, and *instances*.

The *integration* activity consists of research of already existing ontologies which can be reused. Afterwards, *implementation* transforms the previously described model into the desired ontology language. Finally, *evaluation* ensures that the ontology that has been implemented corresponds to its specification.

Important to note about *METHONTOLOGY* is the absence of *documentation* as a separate
139-140 development step. Each of the other activities enforces the creation of one or more artefacts
2 notes: (diagrams or tables) which precisely documents the results of this step. During the implementation activity, all required information about the ontology is taken from these Hence, when all other steps are completed, the documentation is ready as well. Any problems regarding missing, incomplete, or inaccurate documentation are avoided.

141
07.08.13
by: Mario J..

Applications

Applications of *METHONTOLOGY* include the *Legal Ontology* [194], the *Chemical Ontology* [195], the *Graduation Screen Ontology* [196], the *Ontology for Metabolic Pathways* [197], the *Vehicles' Ontology* for checking the consistency of official documents in eGovernment [198], the ontology for a context-aware semantic approach for the effective selection of an assistive software [199] and a cartographic ontology [200]. Some ontology designers opted for an approach that combines *METHONTOLOGY* with another approach, e.g. for the development of an educational ontology [201] using a combination of *METHONTOLOGY* and a *Model Driven Approach* [202]. For the development of an ontology in the hydrographical domain [203], the designers combined the top-down approach of *METHONTOLOGY* with a bottom-up approach based on *Formal Concept Analysis* [173].

Analysis

- **Effort:** Compared to *Ontology 101*, the effort of *METHONTOLOGY* is higher, but far below the effort that arises when applying the software engineering approach from section 4.2.4.

Methodology	Advantage(s)	Disadvantage(s)
Uschold and King	Simple, straight-forward approach	Documentation not enforced Shallow descriptions
TOVE	Evaluation for completeness	Documentation not enforced Tedious formal approach
Ontology 101	Simple and easily comprehensible approach Low effort	Documentation not enforced Seldom used
UPON	Incorporates best practices from software development	Huge development effort
METHONTOLOGY	Widely used Documentation is enforced	-

Figure 4.5: Advantages and disadvantages of the ontology design methodologies discussed in section 4.2.

- **Usage:** *METHONTOLOGY* is one of the most widely used approaches for developing ontologies. This may indicate that it is considered by many ontology developers to be well suited for many domains.

- **Applicability:** There are no limits regarding the domains *METHONTOLOGY* can be used for.

- **Strictness:** *METHONTOLOGY* defines a procedure to follow, but leaves a margin for individual design decisions by the ontology developer.

- **Formality:** This is an informal approach.

- **Level of detail:** The description of this approach is very detailed.

- **Documentation:** *METHONTOLOGY* enforces the generation of documents which document the ontology at the time when the proposed steps are performed; documentation is not a step which has to be performed separately. Hence *METHONTOLOGY* avoids many problems regarding the documentation being in an undesirable state.

142
07.08.13
by: Mario J..

143-145
3 notes:

4.3 Conclusion

The previous section gives detailed insights into some popular methodology of developing ontologies from scratch. Table 4.5 summarises the main advantages and disadvantages of these approaches that have been identified in the previous section.

146
07.08.13
by: Mario J..

Considering the characteristics of the development approaches, *METHONTOLOGY* is chosen for building the *SmartHomeWeather* ontology. The main reasons for that decision are:

147
07.08.13
by: Mario J..

- Compared to other approaches, the effort of following this methodology acceptable.

- The approach of *METHONTOLOGY* enforces the generation of documentation in order to avoid problems regarding missing, incomplete, or inaccurate documentation.

- It is widely used for the development of a variety of ontologies. This implies that many ontology developers consider it suitable for the application on many different domains.

Chapter 5 describes the process of building the *SmartHomeWeather* ontology in detail.

4.4 *METHONTOLOGY*

As section 4.3 opted for *METHONTOLOGY* as the methodology to be used for developing *SmartHomeWeather*, this section presents *METHONTOLOGY* in a more detailed manner.

148-149 2 notes: The inventors of *METHONTOLOGY*, Gómez-Pérez et al., perceived absence of a clear engineering approach towards building ontology from scratch. Hence, they described a process for developing ontologies and specified a life cycle for ontologies. Based upon that, they specified *METHONTOLOGY* as a straight-forward engineering approach for building ontologies [108].

150 07.08.13 by: Mario J.. Several papers describe *METHONTOLOGY* itself [108, 204, 205], while others discuss applications of *METHONTOLOGY* in the development of ontologies (see section 4.2.5).

While the overall approach is the same in all of these articles, there are slight differences regarding the details of each step. For developing the *SmartHomeWeather* ontology in chapter 5, a variant of *METHONTOLOGY* is used that combines aspects from several papers. This methodology is presented in this section. It does not exactly match the methodology presented in any of the papers. 151 07.08.13 by: Mario J..

4.4.1 Ontology development process and life cycle

152 07.08.13 by: Mario J.. The ontology development process used by *METHONTOLOGY* divides ontology development into the following activities that need to be performed [108]:

- **Planification:** This step involves creating a plan regarding which tasks need to be done, how they are arranged and which resources they require.

- 153 07.08.13 by: Mario J.. **Specification:** The purpose, intended uses and end-users are specified in an *Ontologies Requirements Specification Document*.

- **Knowledge acquisition:** Knowledge about the ontology's domain is acquired.

- **Conceptualisation:** The knowledge previously acquired is conceptualised into a conceptual model that describes the problem that shall be solved by the ontology and how the ontology solves it.

- **Formalisation:** This conceptual model is then formalised.

- 154 07.08.13 by: Mario J.. **Integration:** As ontologies are built to be reused, as many existing ontologies as possible are integrated into the new ontology.

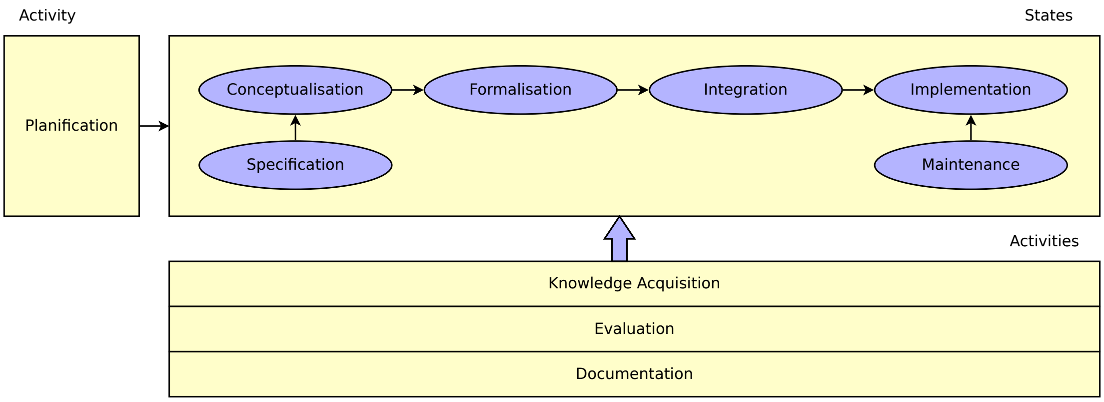

Figure 4.6: States and activities in the life cycle of an ontology according to *METHONTOLOGY* [108]

- **Implementation:** The ontology is then implemented using a formal language.

155
07.08.13
by: Mario J..

- **Evaluation:** Throughout the process of building the ontology, it is continuously evaluated in order to ensure it meets the requirements specified previously.
- **Documentation:** The ontology and all documents belonging to it must be well documented.
- **Maintenance:** It may be necessary to apply modifications throughout the lifetime of the ontology.

156
07.08.13
by: Mario J..

These activities – which are depicted in figure 4.6 – are arranged into the step of *planification* that must be performed at the very beginning of development, a *set of stages* (consisting of *specification*, *conceptualisation*, *formalisation*, *integration*, *implementation* and *maintenance*) through which the ontology moves during its creation and some activities (*knowledge acquisition*, *documentation* and *evaluation*) that are performed throughout the whole development process in parallel to the stages.

Differently to what is shown in figure 4.6, *METHONTOLOGY* follows an evolving life cycle model similar to the iterative-incremental approach that is used in the *Spiral Model* in software development [206]. This life cycle model which allows the ontology to grow according to its needs. Whenever it is necessary, pieces of the ontology can be added, modified and deleted. Thus, one state does not have to be completely finished before the next state is begun. The ontology cycles through each state numerous times until the ontology meets all requirements and the results of each step correspond to each other.

157
07.08.13
by: Mario J..

4.4.2 METHONTOLOGY

This section describes *METHONTOLOGY* as a well-defined approach to perform all activities mentioned above.

For each of the activities, only ideas behind them are covered, but their application is omitted. They are applied in chapter 5 where *METHONTOLOGY* is used to create the *SmartHome-Weather* ontology.

158-159 2 notes: Each section that describes an activity that involves the creation of some documentation artefact (e.g. a table or a document), a template for the respective artefact is presented.

Specification

METHONTOLOGY defines a precise approach for the development of an ontology. It specifies certain activities that need to be performed, how these activities are performed and in which order. Thus, the activity of *planification* is completed by specifying *METHONTOLOGY* itself and the ontology developer is exempted therefrom. Hence, the first step of developing an ontology from scratch is *specification*

During *specification*, an *Ontology Requirements Specification Document* is generated. This document is written in natural language using a set of intermediate representations or using competence questions. It should include

- the name and the purpose of the ontology, its scope, its intended uses, and possible end-users,
- a list of functional requirements (describing the intended functionality of the ontology) and non-functional requirements (describing all intended properties of the ontology not directly related to its functionality), and
- 160-161 2 notes: a list of terms that specify the scope of the ontology.

A good ontology specification document has the following properties:
Figure 4.7 shows a template of an *Ontology Requirements Specification Document* [205].

Knowledge Acquisition

Most of knowledge acquisition is done simultanously with the *specification* phase. It is one of the most important activities and needs to be performed thoroughly as most other activities depend heavily on it.
162 07.08.13 by: Mario J..

Sources of knowledge are experts, books, handbooks, figures, tables and even other ontologies. Knowledge is collected using techniques such as brainstorming, interviews, formal and informal analysis of texts and knowledge acquisition tools.

Conceptualisation

The stage of *conceptualisation* consists of several tasks as shown in figure 4.8 [194]. Again, the figure shows the tasks in a sequential manner. However, as *METHONTOLOGY* uses an evolutional process model, the steps is performed numerous times.
163-164 2 notes:

ONTOLOGY REQUIREMENTS SPECIFICATION DOCUMENT
Name: ...
Purpose: ...
Scope: ...
Implementation language: ...
Intended end-users: ...
Intended uses: ...
Ontology requirements: ...

Non-functional requirements:

- ...

Functional requirements:

- ...

Pre-glossary of terms: ...

Figure 4.7: Template for the *Ontology Requirements Specification Document* of *METHONTOLOGY* [205]

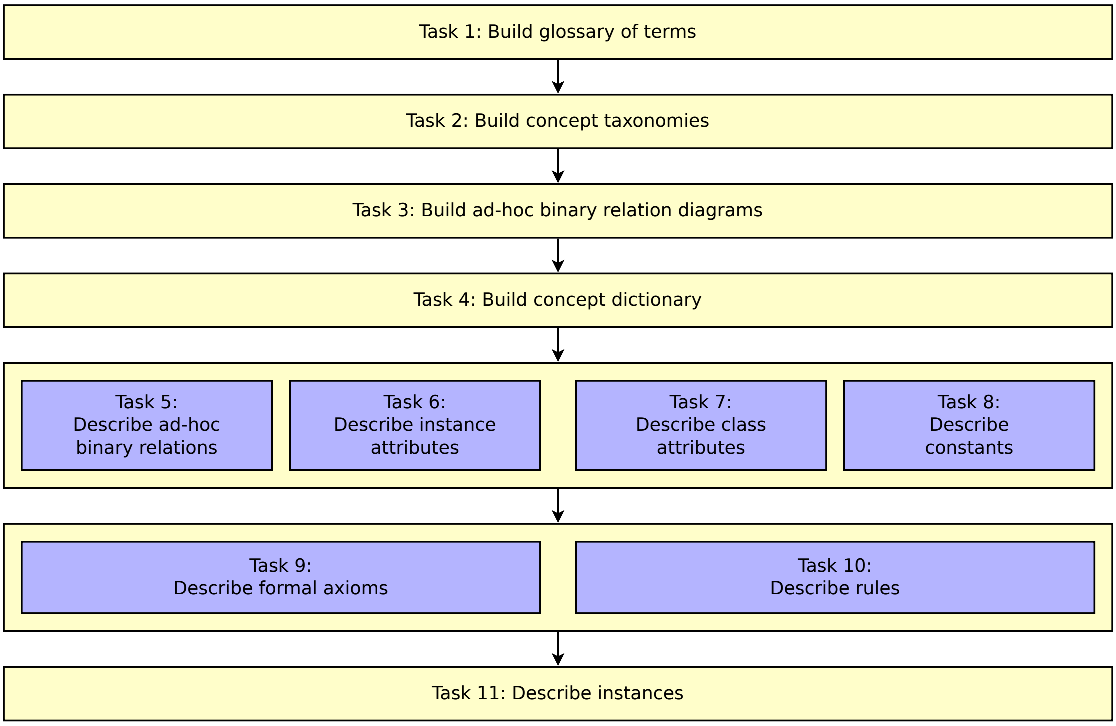

Figure 4.8: Tasks of the conceptualisation activity according to *METHONTOLOGY* [194]

Name	**Acronyms**	**Description**	**Type**
...	...	...	...

Figure 4.9: Template for the glossary of terms as proposed by *METHONTOLOGY*.

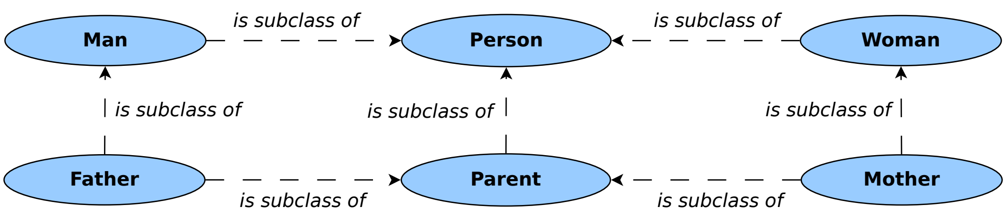

Figure 4.10: Example of a concept-classification tree as proposed by *METHONTOLOGY*.

Task 1: Glossary of Terms At first, the ontologist builds a *Glossary of Terms*. This glossary includes all the relevant terms of the domain (concepts, instances, attributes, relations etc.). It can be built as a table having the columns *name*, *synonyms*, *acronyms*, *description* (for a natural language description of the term) and *type* (specifying whether the term is a concept, an instance, an attribute, a relation etc.).

Figure 4.9 shows a template for the glossary of terms.

Task 2: Concept Taxonomies Once the glossary of terms contains a sizeable number of concepts, these ontologies are arranged in one or more taxonomies that define the concept hierarchy.

METHONTOLOGY proposes the use of four taxonomic relations:

1. **Subclass-Of**: If a concept B is a *Subclass-Of* a concept A, every instance of B is also an instance of A.

2. **Disjoint-Decomposition**: A *Disjoint-Decomposition* of a concept C is a set of subclasses of C such that an instance of one of these subclasses can never be a subclass of another of these subclasses, while an instance of C is not necessarily an instance of one of its subclasses.

3. **Exhaustive-Decomposition**: A *Exhaustive-Decomposition* of a concept C is a set of subclasses of C such that every instance of C is an instance of at least one of its subclasses.

4. **Partition**: A *Partition* of C is a set of subclasses of C such that every instance of C is an instance of exactly one of its subclasses.

165
07.08.13
by: Mario J..

The concept taxonomies are visualised in *concept-classification trees* which are diagrams that depict the concepts and their taxonomic relations. See figure 4.10 for an example of a concept-classification trees. In case the ontology contains a large number of concepts, the tree may be split into several diagrams in order to keep the trees clear.

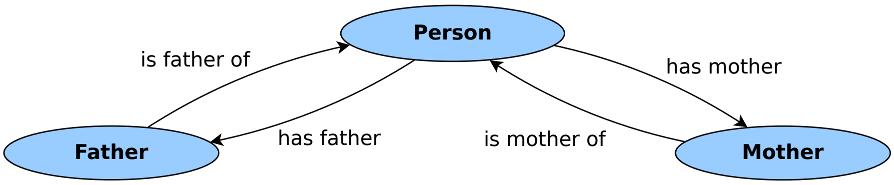

Figure 4.11: Example of a binary relations diagram as proposed by *METHONTOLOGY*.

Name	Instances	Relations
...	...	...

Figure 4.12: Template for the concept dictionary as proposed by *METHONTOLOGY*.

Name	Source concept	Target concept	Maximum source cardinality	Inverse relation
...	...	...	...	...

Figure 4.13: Template for the binary relations table as proposed by *METHONTOLOGY*.

Task 3: Ad-hoc binary relation diagrams In the next step, *ad-hoc binary relation diagrams* are created. This diagrams show all ad-hoc relationships between concepts of the same (or different) concept taxonomy. See figure 4.11 for an example of a binary relation diagram.

Task 4: Concept dictionary The *concept dictionary* contains all domain concepts together with their relations, their instances and their class attributes (i.e. attributes that describe properties of classes) and instance attributes (i.e. attributes that describe properties of instances). All information from the previous steps contribute to this dictionary. The concept dictionary is again being built as a table having appropriate columns for all required information. Like the concept-classification trees, this table may be split into a set of smaller tables if the ontology contains a large number of concepts.

Figure 4.12 shows a template for the concept dictionary.

Task 5: Ad-hoc binary relation details In this step, for all ad-hoc binary relations details are specified in a tabular manner. The resulting table has a row for each relation and columns named *relation name*, *source concept*, *source cardinality (max)*, *target concept*, *inverse relation*. Figure 4.13 shows a template for the binary relations table.

Task 6: Instance attributes This step leads to an *instance attributes table*. That is a table of all *instance attributes* that are listed in the concept dictionary. Each row contains the description of one instance attribute. An instance attribute is an attribute that describes a property of an instance of a concept. Its value may be different for each instance of the concept.

Attribute name	**Concept name**	**Value type**	**Value range**	**Unit**	**Cardinality** (min, max)
...	...	...	...	...	...

Figure 4.14: Template for the instance attributes table as proposed by *METHONTOLOGY*.

Super-concept	**Sub-concept**	**attribute name**	**attribute value(s)**
...	...	...	...

Figure 4.15: Template for the class attributes table as proposed by *METHONTOLOGY*.

Constant name	**Value type**	**Value**	**Measurement unit**
...	...	...	...

Figure 4.16: Template for the constants table as proposed by *METHONTOLOGY*.

The columns of the table are *attribute name*, *concept name*, *value type* (*Integer*, *Float*, *String*, etc.), *value range*, *minimum cardinality* and *maximum cardinality*. Additionally, the following information may be specified: instance attributes, class attributes and constants used to infer values of the attribute; attributes that can be inferred using values of this attribute; formulae or rules that allow inferring values of the attribute; and references used to define the attribute.

See figure 4.14 for a template for the instance attributes table.

Task 7: Class attributes All class attributes that are listed in the concept dictionary are described in detail in the *class attributes table*. Each row describes one class attribute. The columns are *name*, *concept name* (i.e. the name of the concept where the attribute is defined), *value type*, *value(s)*, *minimum cardinality* and *maximum cardinality*. Additionally, all information about related instance attributes, class attributes, constants, rules, and formulae may be specified that are specified in the instance attribute as well.

Figure 4.15 shows a template for the class attributes table.

168
07.08.13
by: Mario J..

Task 8: Constants In this step, the *constants table* is created that specifies details about all the constants listed in the glossary of terms. For each constant is specified by its name, its value type, its value, the measurement unit for numerical constants, and the attributes that can be inferred using the constant.

Figure 4.16 shows a template for the constants table.

169
07.08.13
by: Mario J..

Task 9: Formal axioms In this task, the ontology designer must determine whether the ontology contains formal axioms. In case it contains any, she must describe these axioms precisely in a *formal axioms table*. For each axiom, this table specifies the name, a description in natural language, the logical expression that formally describes the axiom in first-order logic (or the logic

Axiom name	Description	Expression	Referred concepts	Referred attributes	Referred relations	Variables
...	...	...	...	...	...	...

Figure 4.17: Template for the formal axioms table as proposed by *METHONTOLOGY*.

Instance name	Concept name	Attribute	Value(s)
...	...	...	...

Figure 4.18: Template for the instants table as proposed by *METHONTOLOGY*.

the ontology language intended to use is based upon), and all concepts, attributes, relations, and variables that are referred to in the logical expression. Figure 4.17 presents a template for the formal axioms table.

Task 10: Rules Similary to the task of identifying and describing formal axioms within the ontology, in this step the ontology designer must determine whether the ontology contains any rules. If it contains any, she must build a *rules table* to precisely describe all rules and their properties: Their name, their description, an expression in first-order logic, and all concepts, attributes, relations, and variables involved. In contrast to formal axioms, the expression of rules always has the form *if* <*conditions*> *then* <*consequent*>; <*conditions*> is a conjunction of atoms while <*consequent*> is a single atom.

For the rules table, the template for the formal axioms table (see figure 4.17) can be reused.

Task 11: Instances Within an ontology, a set of instances may be predefined. This task involves listing these individuals, again in tabular manner. The columns of this *instances table* are the name of the instance, the name of the concept and all of the instance's attributes together with their respective values. Figure 4.18 depicts a template for the instances table.

Formalisation

Formalisation is the transition from the informal description from the tables and diagrams in the previous step of *conceptualisation* into the chosen ontology language, e.g. *OWL*. As this is tightly coupled with the *implementation* of the ontology (see section 4.4.2), this is a task which is not performed or described separately.

Integration

As ontologies are built for reuse and the wheel shall not be reinvented during the creation of a new ontology, the ontology designer searches for existing ontologies. The goal is to import ontologies that already define terms that are part of the conceptualisation of the ontology currently being developed.

Implementation

175 07.08.13 by: Mario J..

The task of implementing the ontology in an ontology language requires an environment that supports the ontologies selected in the integration step. Features that should be provided by such an environment are [108]

- a lexical and syntactic analyser to guarantee the absence of lexical and syntactic errors,
- an editor for adding, modifying and removing definitions,
- a browser for inspecting the library of ontologies and their definitions,
- a searcher for looking for the most appropriate definitions,
- evaluators for detecting incompleteness, inconsistencies and redundant knowledge and
- an automatic maintainer for managing the inclusion, removal or modification of existing definitions.

176 07.08.13 by: Mario J..

Together with the implementation, the information about the ontology gathered in section 4.4.2 is now formalised into the formal model of the ontology language being used.

In the case of the *SmartHomeWeather* ontology an *OWL* ontology [19] is created using *Protégé* [57] together with the *Pellet* reasoner [58].

Evaluation

177 07.08.13 by: Mario J..

During *Evaluation*, verification takes place whether all artefacts that have yet been created or updated in the previous steps satisfy the requirements that have been specified initially (see section 4.4.2). *Evaluation* is not an activity which is performed at the very end of the development process; instead, *Evaluation* takes place whenever an artefact (a diagram, a table, or the implementation of the ontology) is created or updated in order to ensure that mistakes are found as soon as possible.

178 07.08.13 by: Mario J..

The completed ontology must fulfil all functional and non-functional requirements listed the *Ontology Requirements Specification Document* presented in section 4.4.2. In case of a mismatch, the ontology traverses the activities in the life cycle (*Conceptualisation*, *Formalisation*, *Integration*, *Implementation*, and *Evaluation*) once more.

There may be requirements that an ontology is unable to fulfil due to certain limitations, e.g. the *open world assumption* [53]. For instance, *OWL*, which honors the *open world assumption*, cannot tell the absence of an instance of some concept. This leads to cases where an ontology fails to answer a competency question such as “Does this group only consist of women?”; just because the ontology does not contain an individual which is a man, it does not mean that there is no man; the ontology can only tell that there is no man it knows about.

179-180 2 notes:

Evaluation (and specification) must take limitations into account which affect ontologies generally or the ontology language being used.

Documentation

During the steps described above, a set of documents is compiled. If generated properly and accurately, these documents describe every detail of the ontology. Using this approach, *METHONTOLOGY* forces the ontology designer to document throughout the development process. Any problems that come with documentation being created after the development process – e.g. incomplete or wrong documentation, the ontology designer does not write the documentation herself, or some ideas the designer had in the design process are lost [207] – are avoided.

Hence, in *METHONTOLOGY*, *documentation* is an activity that is not performed explicitly. Once the development process has finished, both the ontology and its documentation are ready to use.

Maintenance

At any time in the future, changes to the ontology may become necessary. A modification of the ontology's requirements may be one reason therefor; inaccurateness that occurs during the ontology development process may be another reason.

Whenever a change is necessary, the ontology again cycles the states of *specification*, *conceptualisation*, *formalisation*, *integration* and *implementation* repeatedly until all requirements are met and all artefacts generated in these states correspond to each other. *Knowledge acquisition* and *evaluation* are again performed throughout all of these states.

CHAPTER 5

The *SmartHomeWeather* ontology

The previous chapters cover all topics that require discussion before being able to build a new ontology from scratch: Chapter 3 discusses all details about weather data that is necessary and reasonable for the *SmartHomeWeather* ontology, what data will be used and where to obtain it. Chapter 2 gives an overview about existing ontologies in the domain of weather data. As none of the existing ontologies being discussed fits the needs of a weather data ontology for smart homes, a new ontology is to be designed. Chapter 4 analyses some of the most popular approaches for building ontologies from scratch. Among those, *METHONTOLOGY* [108] is identified to be the best suitable approach.

Based on these insights, this chapter describes the process of designing the *SmartHome-Weather* ontology in detail. The development process follows the steps proposed by *METHON-TOLOGY* as described in chapter 4.4.2.

5.1 Conventions

All diagrams in this chapter showing parts of an ontology adhere to the following conventions, as seen in the example diagram shown in figure 5.1. These conventions has already been adhered

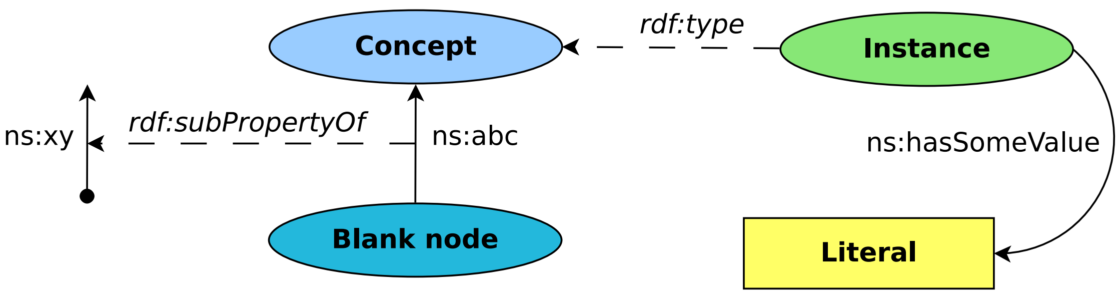

Figure 5.1: Example diagram

to in chapter 2:

- *Concepts* are drawn as ellipses filled with the color `#99CCFF`.
- *Instances* are drawn as ellipses filled with the color `#87E776`.
- *Blank nodes* are drawn as ellipses filled with the color `#23B8DC`.
- *Literals* are drawn as rectangles filled with the color `#FFFF66`.
- *Properties in the* RDF *[42] and* RDFS *[28] namespace* are drawn as dashed lines. Their caption is written in *italics*.
- *Properties in other namespaces* are drawn as solid lines. Their caption is *not* written in *italics*.

Every ontology should stick to a set of naming conventions that are explicitly stated [105]. The conventions for *SmartHomeWeather* are as follows:

- Two concepts, instances and/or properties may not have the same identifier as this is required by *OWL* [19][1] and avoids confusion.
- Two identifiers may not use names that only differ in their capitalization. Using both *Weather State* and *weather state* in the name namespace is possible in *OWL*, but leads to confusion.
- Identifiers may only consist of upper and lower case *ASCII* letters (*A* to *Z* and *a* to *z*), numerical digits from *0* to *9* and spaces, i.e. all identifiers must match the regular expression `^[A-za-z0-9 ]+$`.
- *Concepts* have an identifier that is in singular case and starts with an upper case letter. Typically a concept's identifier is a noun, e.g. *Weather state* or *Weather report*.
- *Properties* have an identifier that starts with a lower case letter and starts with the prefix *has* or *belongs to*, followed by the name of the name of the concept which is the property's *range*. The inverse property of a property having an identifier starting with *has* has an identifier starting with *belongs to*, followed by the inverse property's *range*, and vice versa. As an alternative to the prefix *belongs to*, the prefix *is* in conjunction with the suffix *of* and the inverse property's *domain* may be used.

  186
  07.08.13
  by: Mario J..

  E.g. the name of a property with the domain *Weather report* and the range *Weather state* has the name *has weather state*. If *has weather state* has an inverse property, it will have the name *belongs to weather report* or *is weather state of*.

[1] Using one identifier for more than one concept, instance and properties is possible in *OWL* if an adequate number of namespaces is used. However, *SmartHomeWeather* uses a single namespace.

5.2 Specification

Specification, the first step proposed by *METHONTOLOGY*, aims at creating an *Ontology Requirements Specification Document* using natural language. It adheres to the approach discussed in chapter 4.4.2 and uses the document template taken presented in section 4.4.2 [205].

ONTOLOGY REQUIREMENTS SPECIFICATION DOCUMENT

Name: *SmartHomeWeather*

Purpose: The ontology covers data about weather phenomena occurring at a certain location somewhere on Earth between the present and 24 hours in the future. Weather data will be acquired from both Internet services as well as from weather sensors mounted at the desired location. This weather data will enable a smart home system using *SmartHomeWeather* to make decisions based on current and future weather conditions.

Scope: The ontology has to cover a set of five core concepts from the domain of weather data:

- *weather phenomenon*: Represents a certain weather element. Relevant weather elements are *temperature*, *humidity*, *dew point*, *wind speed* and *direction*, *precipitation intensity* and *probability*, *atmospheric pressure*, *cloud cover*, *solar radiation*, and the *sun's position*.
- *Weather condition*: Overall state of the weather given by a simple verbal description: *sun*, *light clouds*, *partly cloudy*, *cloudy*, *fog*, *rain*, *snow*, *sleet*, *thunder*.
- *Weather state*: Summarizes all weather phenomena for a certain time.
- *Weather report*: Summarizes all data acquired at a certain time about the current weather or the weather some time in the future. Exactly one *weather state* is linked to each *weather report*.
- *Weather report source*: Source where the data belonging to a *weather report* has been obtained from (either an Internet weather service or a local weather sensor).

Implementation language: The ontology is implemented in OWL *2* [19] using *Protégé* [57] and the *Pellet reasoner* [58].

187 07.08.13 by: Mario J..

Intended end-users: End-users of the ontologies are inhabitants of smart homes.

188 07.08.13 by: Mario J..

Intended uses: The ontology shall provide knowledge to ontology-based smart home systems about the current and future state of the weather in order to enable the system to make decisions based on that knowledge.

Ontology requirements:

Non-functional requirements:

- The ontology must adhere to the naming conventions presented in section 5.1 regarding the identifiers that may be used for classes, properties and individuals.
- The ontology must be documented thoroughly in order to make it easily reusable.
- The ontology must re-use existing ontologies wherever possible.

Functional requirements: The functional requirements are covered by the competency questions that the ontology shall be able to answer (see section 3.1):

- What is the current weather situation?
- What will the weather situation be in one hour, in two hours, . . . , in 24 hours?
- What is the current temperature, humidity, wind speed, . . . ?
- What will be the temperature, humidity, wind speed, . . . in one hour, in two hours, . . . , in 24 hours?
- What will be the minimum temperature, humidity, . . . over the next 24 hours? What about maximum values?
- Will the weather change? Will the temperature, humidity, . . . rise or fall?
- Does it rain? Will it rain in the next hours? Will it rain today?
- Will there be sunshine today?
- Do we need to irrigate the garden?
- Will there be severe weather?
- Will temperature drop/stay below $0^\circ C$?
- When can we open windows and when do we have to keep them shut?
- When do we need sun protection?
- When will it outside be colder than inside the house? When will it be warmer?

189
07.08.13
by: Mario J..

Pre-glossary of terms: These are all terms that can be extracted from the competency questions, in alphabetical order:
24 hours, airing, current weather, frost, future weather, humidity, humidity rise, humidity fall, irrigation, minimum, maximum, rain, room temperature, severe weather, sunshine, sun protection, temperature, temperature rise, temperature fall, weather change, wind speed.

In the following sections, the *SmartHomeWeather* ontology is built in a way to meet all above requirements, if possible. Section 5.7 evaluates if the resulting ontology fits the specification and which shortcomings the ontology comes with.

5.3 Knowledge Acquisition

The second step proposed by *METHONTOLOGY* is *Knowledge Acquisition*. All knowledge required to build the *SmartHomeWeather* ontology is presented in the chapters 2 and 3. These chapters discuss in detail:

- Which weather data are relevant for smart homes?
- Which weather data are available from sensors (section 3.2) and Internet services (section 3.3.1)? How can this data be acquired?
- Which data do not have any use for *SmartHomeWeather* due to being too complicated or because they cannot be processed in an ontology in a useful way?

190 07.08.13 by: Mario J..

- What knowledge about weather in general is required to build an appropriate ontology (refer to chapter 3?

Furthermore, chapter 2 covers existing ontologies that cover the domain of weather data. An additional source of knowledge is available through works about weather in general, e.g. the *Glossary of Meteorology* [111] by the *American Meteorological Society* [208].

5.4 Conceptualisation

In the third step of *METHONTOLOGY*, the *Conceptualisation* step, the domain knowledge is structured into a conceptional model that describes the problem and its solution in terms of the domain vocabulary that has been identified in the *Specification* process.

The starting point of *Conceptualisation* is a complete *Glossary of Terms* that covers all concepts, instances, attributes and binary relations that will form the ontology. Besides the glossary, this section covers *Concept-classification trees* (section 5.4.2), *Binary relationship diagrams* (section 5.4.3), *Concept dictionaries* (section 5.4.3), *Binary relations tables* (section 5.4.5), *Instance attribute tables* (section 5.4.5) and *Class attributes tables* (section 5.4.7). *Constant tables*, *Formal axiom tables* and *Rules tables* have been omitted as the components described by these tables do not appear in *SmartHomeWeather*.

In this section, only those deliverables are presented that are necessary to fully understand the structure of *SmartHomeWeather*. All tables that have only been created for the sake of completeness can be found in appendix A.1.

5.4.1 Glossary of Terms

When describing the scope of *SmartHomeWeather*, the *Ontology Requirements Specification Document* in section 5.2 mentions five top-level concepts (i.e. concepts that do not have a superclass except *Thing* in *OWL*) of the ontology: *Weather report*, *Weather state*, *Weather phenomenon*, and *Weather condition*. All other concepts are sub-concepts of these five concepts.

191-192
2 notes:

In this section, only a list of terms is given; the complete *Glossary of Terms* with short descriptions of each term can be found in the appendix B which starts on page 113.

Concepts: Section 3.4 presents the weather elements that are used in *SmartHomeWeather*. In the ontology, weather elements are represented by concepts that are sub-concepts of *Weather phenomenon*, e.g. there is a concept *Temperature* for measurements of temperature, or *Humidity* for measurements of relative humidity.

For all weather elements except *Dew point*, categories are introduced in order to allow easy differentiation of weather observations by their respective measurement values. In the case of *Temperature*, the sub-concepts differ from each other by the observed temperature values. The sub-concepts of *Temperature* are *Frost* (for an observed temperature value of below 0 °C), *Cold* (at least 0 °C and less than 10 °C), *Below room temperature* (at least 10 °C and less than 20 °C), *Cold* (at least 20 °C and at most 25 °C), *Above room temperature* (more than 25 °C and at most 30 °C), and *Heat* (more than 30 °C). Refer to section 5.4.2 for the concept-classification trees that result from this approach including the definitions of the respective sub-concepts.

193
07.08.13
by: Mario J..

A *weather report* can encapsulate data either about the current weather or about the weather some time in the future which is specified by its *start time*. Additionally, weather data can origin at a set of weather sensors or at an Internet weather service. To take this into account, a few sub-concepts of *weather report* are introduced:

If the *weather report* describes the current weather, it is a *Current weather report*; if it describes the future weather, it is a *Forecast weather report*. Depending on how far the *Weather report*'s *start time* lies ahead, it is a *Short range weather report* (at most 3 hours in the future), a *Medium range weather report* (more than 3 hours and less than 12 hours in the future) or a *Long range weather report* (at least 12 hours in the future). Furthermore, there are sub-concepts of *weather report*, each having a *start time* of 1, 2, 3, 6, 9, 12, 15, 18, or 24 hours; these concepts are named *Forecast 1 hour weather report*, *Forecast 2 hours weather report* etc., respectively.

If the source of weather data is a *Sensor source*, the corresponding *Weather report* is a *Weather report from sensor*; otherwise (if the source of weather data is a *Service source*), it is a *Weather report from service*.

A *Weather report* that is both a *Current weather report* and a *Weather report from sensor* is a *Current weather report from sensor*. A *Weather report* that is both a *Current weather report* and a *Weather report from service* is a *Current weather report from service*.

Several sub-concepts of *Weather state* describe certain combinations of instances of *Weather phenomenon* being associated with a *Weather state*. These concepts are listed below; refer to the glossary in appendix B for their definitions.

194
07.08.13
by: Mario J..

Thus, these are the concepts that can be found in *SmartHomeWeather* are:

- *Weather condition*
- *Weather phenomenon*:
  - *Atmospheric pressure*: *Very low pressure*, *Low pressure*, *Average pressure*, *High pressure*, *Very high pressure*

- *Cloud cover*: *Clear sky*, *Partly cloudy*, *Mostly cloudy*, *Overcast*, *Unknown cloud cover*.
- *Dew point*.
- *Humidity*: *Very dry*, *Dry*, *Normal humidity*, *Moist*, *Very moist*.
- *Precipitation*: *No rain*, *Light rain*, *Medium rain*, *Heavy rain*, *Extremely heavy rain*, *Tropical storm rain*.
- *Solar radiation*: *No radiation*, *Low radiation*, *Medium radiation*, *High radiation*, *Very high radiation*.
- *Sun position*: *Day*, *Solar twilight*, *Sun below horizon*, *Twilight*, *Civil twilight*, *Nautical twilight*, *Astronomical twilight*, *Night*, *Sun from north*, *Sun from east*, *Sun from south*, *Sun from west*.
- *Temperature*: *Frost*, *Cold*, *Below room temperature*, *Room temperature*, *Above room temperature*, *Heat*.
- *Wind*: *Directional wind*, *North wind*, *East wind*, *South wind*, *Wind wind*, *Calm*, *Light wind*, *Strong wind*, *Storm*, *Hurricane*.

- *Weather report*: *Weather report from sensor*, *Weather report from service*, *Current weather report*, *Current weather report from sensor*, *Current weather report from service*, *Forecast weather report*, *Short range weather report*, *Medium range weather report*, *Long range weather report*, *Forecast 1 hour weather report*, *Forecast 2 hours weather report*, ..., *Forecast 24 hours weather report*

- *Weather source*: *Sensor source*, *Service source*.

- *Weather state*: *airing weather*, *calm weather*, *clear weather*, *cloudy weather*, *cold weather*, *dry weather*, *fair weather*, *hot weather*, *moist weather*, *no rain weather*, *pleasant temperature weather*, *rainy weather*, *severe weather*, *sun protection weather*, *thunderstorm*, *very rainy weather*, *windy weather*

Relations: Instances of the concepts are associated to each other with binary relations, which are:

- *has source* and *is source of* which connect instances of *Weather report* and *Weather source*.

- *has weather state* and *belongs to weather report* which connect instances of *Weather report* and *Weather state*.

- *has condition* which connects instances of *Weather state* and *Weather condition*.

- *has weather phenomenon* and *belongs to state* which connect instances of *Weather state* and *Weather phenomenon*.

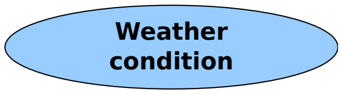

Figure 5.2: Concept-classification tree for *Weather condition*

- *has previous weather state* and *has next weather state* which connect two instances of *Weather state*.

The following relations link instances of concepts from other ontologies than *SmartHomeWeather* to instances of concepts inside the ontology: *has start time*, *has end time*, *has observation time*, and *location*.

The only data property in *SmartHomeWeather* is *has priority* which specifies an integer value indicating which *Weather report* for a certain period of time is to be preferred over another *Weather report* for the same period of time.

Individuals: The only predefined individuals are instances of the concept *Weather condition*; they represent the overall state of the weather for a certain *Weather state*. These individuals are: *cloud*, *fog*, *light clouds*, *partly cloudy*, *rain*, *sleet*, *snow*, *sun*, and *thunder*.

5.4.2 Concept-classification trees

195 07.08.13 by: Mario J.. As stated in section 5.4.1, there are fix top-level concepts; all other concepts are sub-concepts to these top-level concepts. As a consequence, each of these concepts becomes root of a tree of concepts. These *Concept-classification trees* are presented in this section.

196 07.08.13 by: Mario J.. A *Weather condition* does not have any sub-concepts. Hence, its classification tree which is shown in figure 5.2 looks rather simple.

A *Weather phenomenon* represents a certain weather element. Every specific weather element is a sub-concept of *Weather phenomenon*; the evolving tree is shown in 5.3. For sake of clarity, this tree is broken up to several diagrams; all sub-concepts of sub-concepts of *Weather phenomenon* are not shown in 5.3. There is a separate diagram for each sub-concept of *Weather phenomenon*:

- *Atmospheric pressure* (figure 5.4): Depending on the pressure value, an instance of *Atmospheric pressure* is an instance of exactly one of its sub-concepts: *Very low pressure* (pressure value below 998 hPa), *Low pressure* (at least 998 hPa and less than 1008 hPa), *Average pressure* (at least 1008 hPa and less than 1018 hPa), *High pressure* (at least 1018 hPa and less than 1028 hPa), and *Very high pressure* (at least 1028 hPa).

- 197 07.08.13 by: Mario J.. *Cloud cover* (figure 5.5): The sub-concepts of *Cloud cover* are defined using different cloud coverage values: *Clear sky* (cloud coverage of 0 *okta*), *Partly cloudy* (1 *okta* to 4 *okta*), *Mostly cloudy* (5 *okta* to 7 *okta*), *Overcast* (8 *okta*) and *Unknown cloud cover* (9 *okta*).

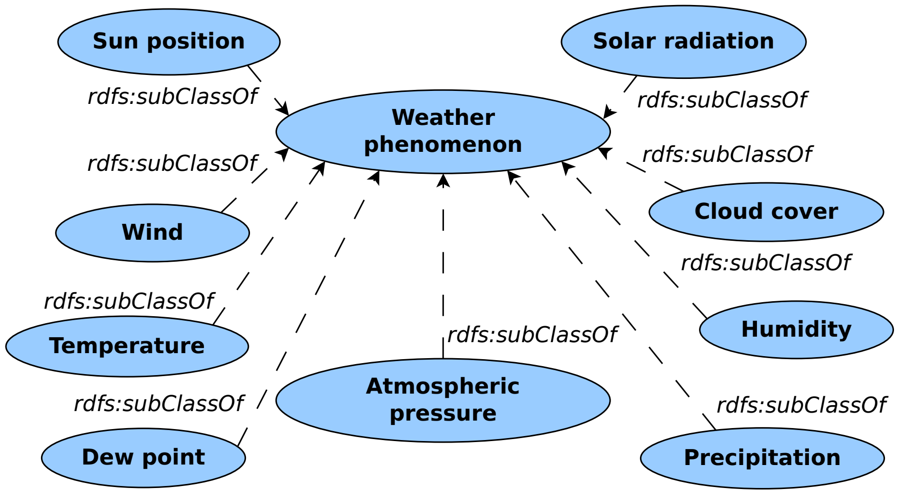

Figure 5.3: Concept-classification tree for *Weather phenomenon*

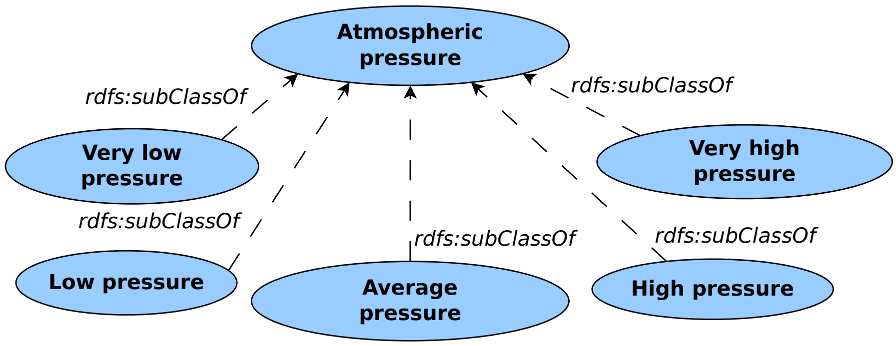

Figure 5.4: Concept-classification tree for *Atmospheric pressure*

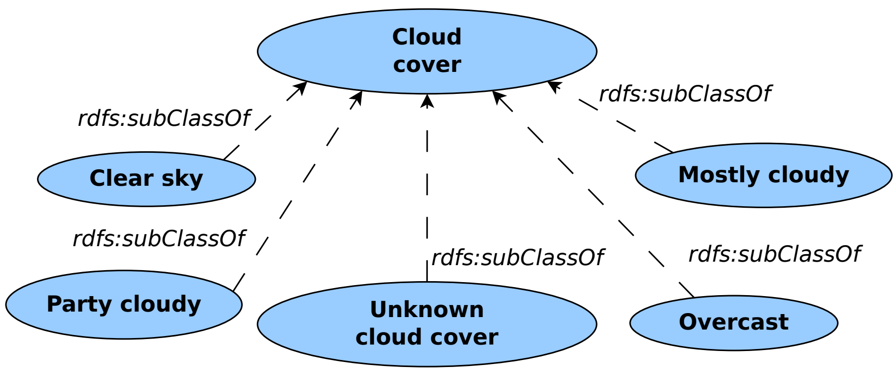

Figure 5.5: Concept-classification tree for *Cloud cover*

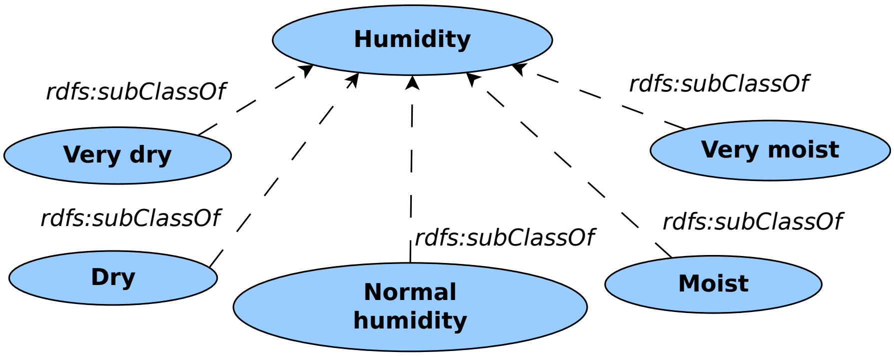

Figure 5.6: Concept-classification tree for *Humidity*

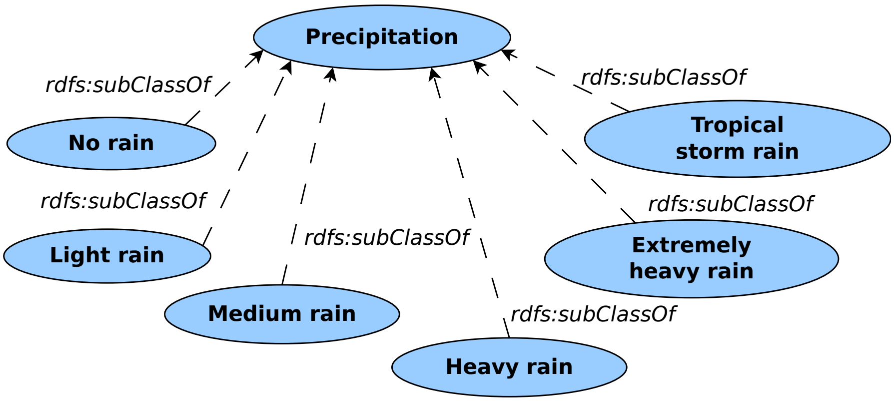

Figure 5.7: Concept-classification tree for *Precipitation*

- *Humidity* (figure 5.6): The sub-concepts of *Humidity* are *Very dry* (value of relative humidity of less than 30 percent), *Dry* (at least 30 percent and less than 40 percent), *Normal humidity* (at least 40 percent and at most 70 percent), *Moist* (more than 70 percent and at most 80 percent), and *Very moist* (more than 80 percent).

- *Precipitation* (figure 5.7): The sub-concepts of *Precipitation* are *No rain* (precipitation intensity of 0 mm/h), *Light rain* (precipitation intensity of more than 0 mm/h and at most 5 mm/h), *Medium rain* (precipitation intensity of more than 5 mm/h and at most 20 mm/h), *Heavy rain* (precipitation intensity of more than 20 mm/h and at most 50 mm/h), *Extremely heavy rain* (precipitation intensity of more than 50 mm/h and at most 100 mm/h), and *Tropical storm rain* (precipitation intensity of more than 100 mm/h). For *No rain*, the precipitation probability is 0; for the other sub-concepts, the precipitation probability is greater than 0.

198
07.08.13
by: Mario J..

- *Solar radiation* (figure 5.8): In this case, the sub-concepts are *No radiation* (solar radiation value of 0 W/m^2), *Low radiation* (solar radiation value of more than 0 W/m^2 and less than 250 W/m^2), *Medium radiation* (solar radiation value of more than 250 W/m^2 and less than

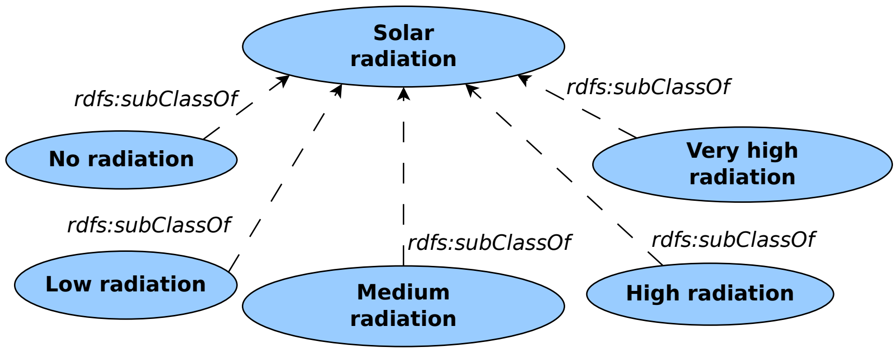

Figure 5.8: Concept-classification tree for *Solar radiation*

$500\,\mathrm{W/m^2}$), *High radiation* (solar radiation value of more than $500\,\mathrm{W/m^2}$ and less than $750\,\mathrm{W/m^2}$), *Very high radiation* (solar radiation value of more than $750\,\mathrm{W/m^2}$).

- *Sun position* (figure 5.9): There are sub-concepts that are defined depending on the sun's azimuth angle, while others are defined depending on the sun's elevation above horizon:
  - Those depending on the azimuth angle are *Sun from north* (azimuth angle of at least $0°$ and at most $45°$ or of more than $315°$ and less than $360°$), *Sun from east* (azimuth angle of more than $45°$ and at most $135°$), *Sun from south* (azimuth angle of more than $135°$ and at most $225°$), and *Sun from east* (azimuth angle of more than $225°$ and at most $315°$).
  - The sub-concepts defined via the elevation angle are *Day* (elevation angle of at least $0°$ and at most $90°$), *Solar twilight* (elevation angle of at least $0°$ and less than $6°$), *Sun below horizon* (elevation angle of at least $-90°$ and less than $0°$), *Twilight* (elevation angle of at least $-18°$ and less than $0°$), *Civil twilight* (elevation angle of at least $-6°$ and less than $0°$), *Nautical twilight* (elevation angle of at least $-12°$ and less than $-6°$), *Astronomical twilight* (elevation angle of at least $-18°$ and less than $-12°$), and *Night* (elevation angle of at least $-90°$ and less than $-18°$). Refer to the *Glossary of Meteorology* for the definitions of the different states of twilight [111].

- *Temperature* (figure 5.10): The sub-concepts of *Temperature* are *Frost* (temperature value below $0\,°\mathrm{C}$), *Cold* (at least $0\,°\mathrm{C}$ and less than $10\,°\mathrm{C}$), *Below room temperature* (at least $10\,°\mathrm{C}$ and less than $20\,°\mathrm{C}$), *Room temperature* (at least $20\,°\mathrm{C}$ and at most $25\,°\mathrm{C}$), *Above room temperature* (more than $25\,°\mathrm{C}$ and at most $30\,°\mathrm{C}$), and *Heat* (more than $30\,°\mathrm{C}$).

- *Wind* (figure 5.11): As for *Sun position*, there are two types of sub-concepts of *Wind* – those defined by the wind speed and those defined by the wind direction:
  - If an instance of *Wind* has the property *has wind direction*, it is defined to be an instance of *Directional wind*. This concept in turn has four sub-concepts: *North wind* (wind direction of at least $0°$ and less than $45°$ or of at least $315°$ and less than

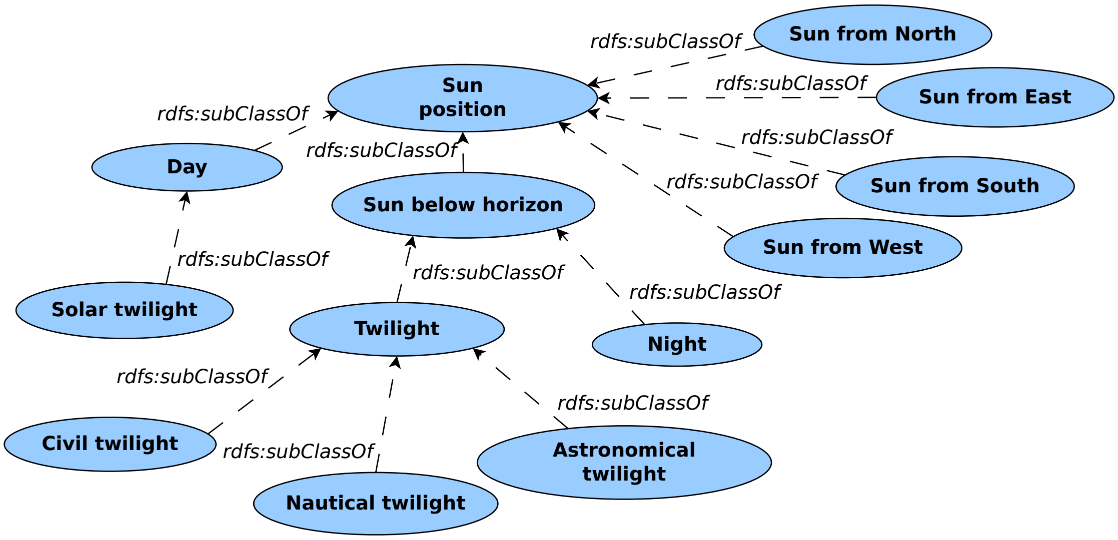

Figure 5.9: Concept-classification tree for *Sun position*

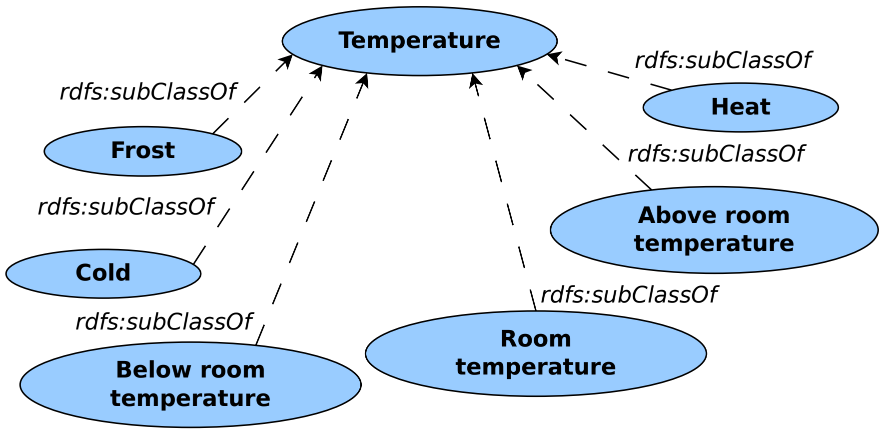

Figure 5.10: Concept-classification tree for *Temperature*

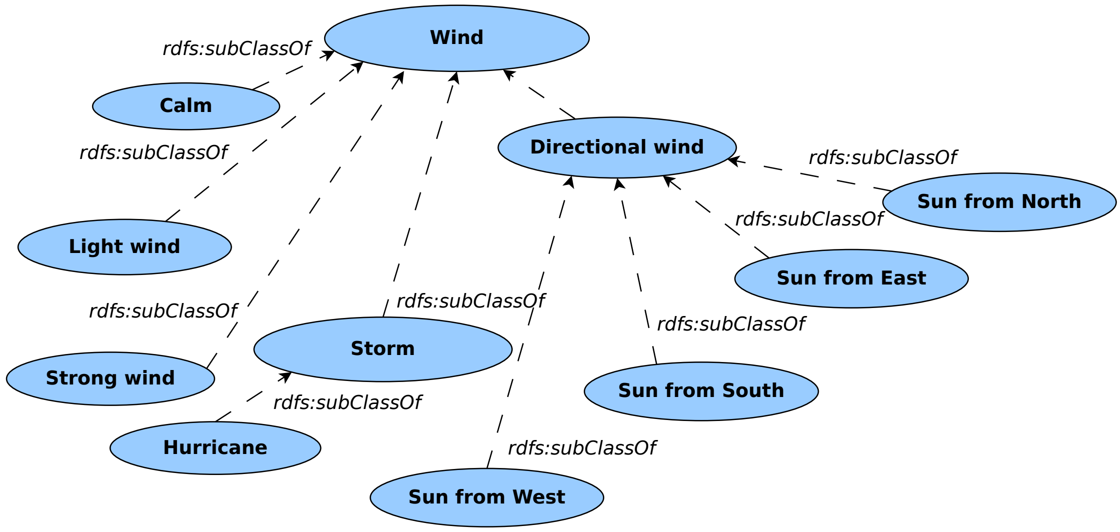

Figure 5.11: Concept-classification tree for *Wind*

360°), *East wind* (at least 45° and less than 135°), *South wind* (at least 135° and less than 225°), and *East wind* (at least 225° and less than 315°).

– Depending on the wind speed, there are the sub-concepts *calm* (wind speed of at least 0 km/h and less than 1 km/h), *light wind* (at least 1 km/h and less than 10 km/h), *strong wind* (at least 10 km/h and less than 20 km/h), *storm* (at least 20 km/h), and *hurricane* (at least 32 km/h).

There is no separate diagram for *Dew point* as that concept does not have any sub-concepts; hence its concept-classification tree consists of a single node.

Refer to section A.1 in the appendix for a tabular display of the sub-concepts of *Weather phenomenon*.

A *Weather report* has two attributes that define its main characteristics: *has start time* and *has source*. As discussed in section 5.4.1, a number of sub-concepts is defined in order to reflect different values of these two attributes. The resulting concept-classification tree is shown in figure 5.12.

The concepts *Short range weather report*, *Medium range weather report* and *Long range weather report* each do have sub-concepts which have been omitted from the above diagram for clarity. These sub-concepts are:

- *Short range weather report*: *Forecast 1 hour weather report*, *Forecast 2 hours weather report* and *Forecast 3 hours weather report* for weather reports describing the weather in one, two and three hours, respectively.

- *Medium range weather report*: *Forecast 6 hour weather report* and *Forecast 9 hours weather report* for weather reports describing the weather in 6 and 9 hours, respectively.

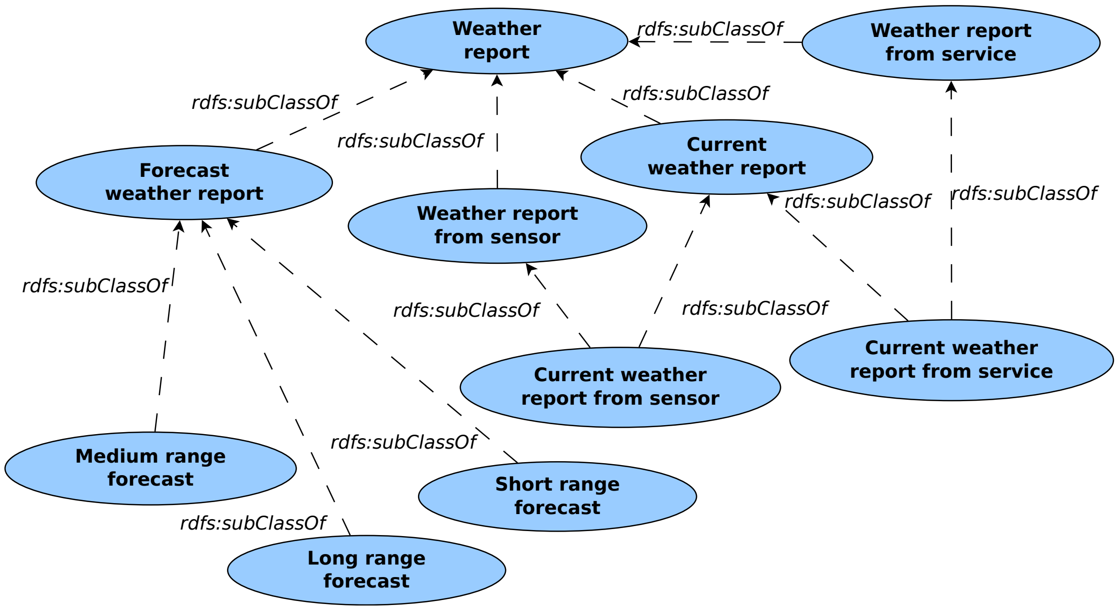

Figure 5.12: Concept-classification tree for *Weather report*

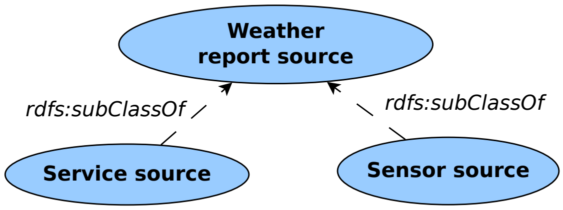

Figure 5.13: Concept-classification tree for *Weather source*

- *Long range weather report*: *Forecast 12 hour weather report*, *Forecast 15 hours weather report*, *Forecast 18 hours weather report*, *Forecast 21 hours weather report* and *Forecast 24 hours weather report* for weather reports describing the weather in 12, 15, 18, 21 and 24 hours, respectively.

A *Weather source* can either be a *Sensor source* or a *Service source* (see figure 5.13).

A *Weather state* represents the set of weather phenomena that belong to a certain *Weather report*. In order to emphasise certain combinations of instances of *Weather phenomenon* being linked to the same instance of *Weather state*, several sub-concepts of *Weather state* are introduced (see section 5.4.1). The resulting tree is seen in figure 5.14.

5.4.3 Binary relations diagram

The purpose of a *Binary relations diagram* is to present all binary relations between concepts in the ontology (see figure 5.15).

199
07.08.13
by: Mario J..

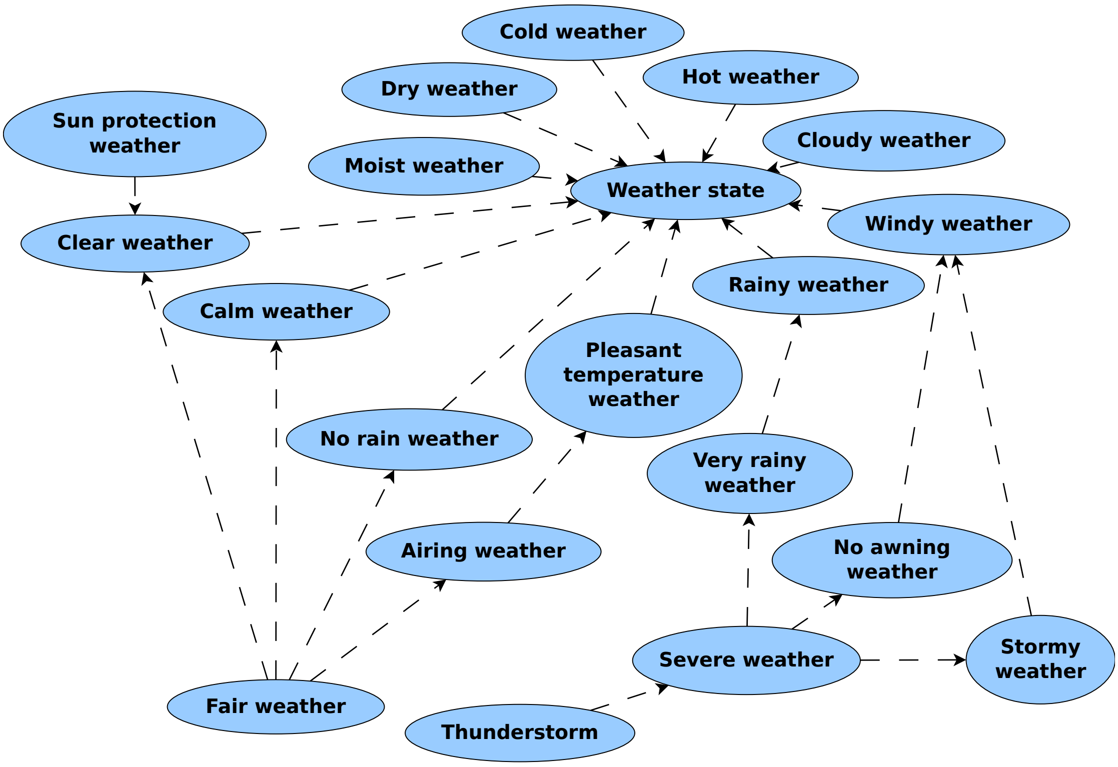

Figure 5.14: Concept-classification tree for *Weather state*. All properties are of type *rdfs:subClassOf*

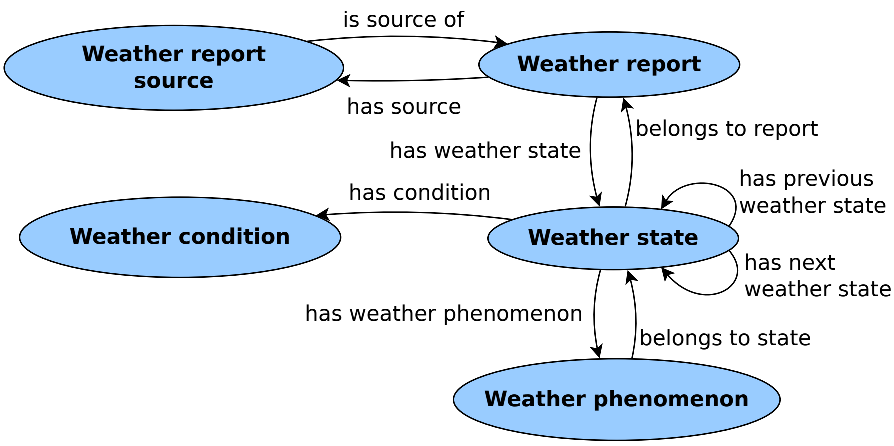

Figure 5.15: Binary relations diagram of *SmartHomeWeather*

5.4.4 Concept dictionaries

A *Concept dictionary* lists all concepts together with their names, instances, class attributes, instance attributes, and relations. For the sake of clarity, this table is split up into several tables, one for each of the concept-classification trees from section 5.4.2.

The tables can be found in the appendix in figure A.1 (*Weather condition*, *Weather report*, and *Weather source*) and figure A.2 (*Weather phenomenon* and *Weather state*). In these tables, sub-concepts having the same instances, class attributes, instance attributes and relations as their super-concepts are omitted. Furthermore, any columns that are not filled with any content in any row are omitted.

5.4.5 Binary relations table

The *Binary relation table* specifies all relations from section 5.4.3 in detail. This includes the relations' names, their source and target concepts, their maximum source cardinalities, and their inverse relations, if any. The table can be found in the appendix in figure A.3.

5.4.6 Instance attributes table

An *Instance attributes table* lists all instance attributes in *SmartHomeWeather* together with the concept where they belong to, their value type and value range, the unit of measurement and their cardinality. The *Instance attributes table* can be found in the appendix in figure A.4.

200
07.08.13
by: Mario J..

The data types *xsd:integer* and *xsd:decimal* in the table refer to the types defined in XML *Schema* [45]. For sake of simplicity, this table does not respect use the *Measurements Unit Ontology* (*MUO*) [97]. In the ontology, all attributes listed below that belong to a sub-concept of *Weather phenomenon* do not refer to a literal as stated here. Instead, they link to a blank node of type *Quality value* which in turn has two attributes, one for the literal value (named *numerical value*) and one for the unit (named *measured in*). The type given in the table below is the type of the value the property *numerical value* refers to. See section 5.6.1 for details about the implementation of *MUO* in *SmartHomeWeather*.

5.4.7 Class attributes table

Many concepts within the *SmartHomeWeather* ontology define themselves to be specializations of other concepts (see section 5.4.2 above). E.g. the concepts *Very low pressure*, *Low pressure*, *Average pressure*, *High pressure* and *Very high pressure* are all sub-concepts of *Atmospheric pressure*. They all differ by the value of the instance attribute *has pressure value* that every instance of *Atmospheric pressure* has.

These concepts are summarized in the *Class attributes table* in the appendix in figure A.5, figure A.6, and figure A.7. As in the *Instance attributes tables* in section 5.4.6, this table does not respect the use of *MUO*; no units are specified.

5.4.8 Instances table

The only pre-defined instances in *SmartHomeWeather* are the instances of the concept *Weather condition*. Their details can be found in the *Instance table* in the appendix in figure A.9.

5.5 Integration

One of the goals when designing an ontology is to reuse existing ontologies where possible [22, 23]. Chapter 2 sheds some light on a selection of existing ontologies around the domain of weather data. In the domain of *SmartHomeWeather*, four areas have been identified where existing ontologies may be reused. These areas are location data (section 2.3.1), measurement units (section 2.3.3), specifications of date and time (section 2.3.2) and weather concepts (section 2.2).

The following ontologies have been selected for the import into *SmartHomeWeather*:

- *OWL-Time* [95] is used for specifying temporal data (for date and time of a *Weather report*).
- The *Basic Geo (WGS84 lat/long) Vocabulary* [91] is used for specifying the location a *Weather report* if valid for.
- The *Measurement Units Ontology* [97] (*MUO*) is used to enrich measurement values of each *Weather phenomenon* with a unit (e.g. temperature in °C, rain in mm/h). Although *MUO* does come with a few shortcomings (see section 5.6), it is the ontology that has been identified to be the the one that fits *SmartHomeWeather*'s requirements best.
- Unfortunately, no weather ontology has been identified that defines weather concepts in a way that suits *SmartHomeWeather*'s requirements. Hence, *SmartHomeWeather* defines its own concepts and properties for the domain of weather data.

Section 5.6 describes the reuse of the aforementioned ontologies within the *SmartHome-Weather* ontology in detail.

201
07.08.13
by: Mario J..

5.6 Implementation

After exhaustive analysis and structuring in the previous sections, the step of implementing the ontology has become a straight-forward task.

The *SmartHomeWeather* ontology is implemented in OWL using *Protégé* 4.1 together with the *Pellet OWL* 2 Reasoner. *Pellet* includes *Pellint* [209], an ontology performance tool that uses a set of patterns to find possible performance problems in an *OWL* ontology. *Pellint* has been used intensively to ensure it does not report any problems that could affect reasoning performance.

To ensure that the reasoning of all sub-concepts of *weather phenomenon*, *weather state* and *weather report* works correctly, *JUnit* [210] is used (see section 6.3). Every test case loads the *SmartHomeWeather* ontology using the *Jena framework*, adds appropriate individuals and checks using the *Pellet reasoner* if reasoning is performed in the desired manner.

During implementation of *SmartHomeWeather* in *OWL*, the identifiers of concepts, properties and instances in this document are modified by removing all space characters and concatenating all parts of the identifier using *camel case* [211], e.g. *Weather state* becomes *WeatherState* and *has start time* becomes *hasStartTime*. Hence, in the *OWL* implementation, all identifiers match the regular expression `^[A-za-z0-9]+$`.

5.6.1 Imported ontologies

During the implementation step, the ontologies listed in section 5.5 need to be imported and integrated. Their use necessitates the application of certain patterns required by these ontologies.

OWL-Time [95] defines the concept *Temporal entity* and its sub-concepts *Instant* and *Interval*, both having a self-explanatory name. The properties *has start time*, *has end time* and *has observation time* of *weather report* link to instants; however, only *has observation time* is implemented to link an instance of *Instant* to a *Weather report*. For *has start time* and *has end time*, instances of *Interval* are used. Such an instance represents the interval between the *observation time* and the time the *Weather report* is valid from/until (in hours). This simplifies reasoning of the sub-concepts of *Weather report* which depend on the report's *start time*; e.g. an instance of *Forecast 1 hour weather report* can be defined as an instance of *Weather report* having a *start time* of 1 (hours).

202
07.08.13
by: Mario J..

Figure 5.16 shows a *Weather report* together with its with *start time* and *end time*; figure 5.17 displays a *Weather report* together with its *observation time*. The *Time Zone Ontology* that comes with *OWL-Time* is not used by *SmartHomeWeather*. All times are given in *UTC*.

Figure 5.18 shows how an instance of *Temperature* would be implemented if no ontology for units of measurements would be used. With the introduction of the *Measurement Units Ontology* (*MUO*), the data property and the literal are removed; an object property that is a sub-property of *Quality value* (which is defined by *MUO*) takes the place of the data property. It links to a blank node which in turn has two properties: *Measured in* and *numericalValue*. The property *Measured in* is an object property that refers to the unit being used which is represented by an instance of *MUO*'s concept *Unit of measurement*. Using the data property *numericalValue*, the literal is connected to the blank node. The resulting pattern is seen in figure 5.19.

MUO is an ontology that is easy to implement in an already-existing ontology. Its major drawback in the case of *SmartHomeWeather* is that it affects reasoning time negatively. Repeated tests showed that reasoning time increased by about 30% when introducing *MUO*. However, the slowdown caused by *MUO* is accepted in favour of the usage of units of measurement.

203
07.08.13
by: Mario J..

Figure 5.20 shows how the *location* of a *Weather report* is encoded.

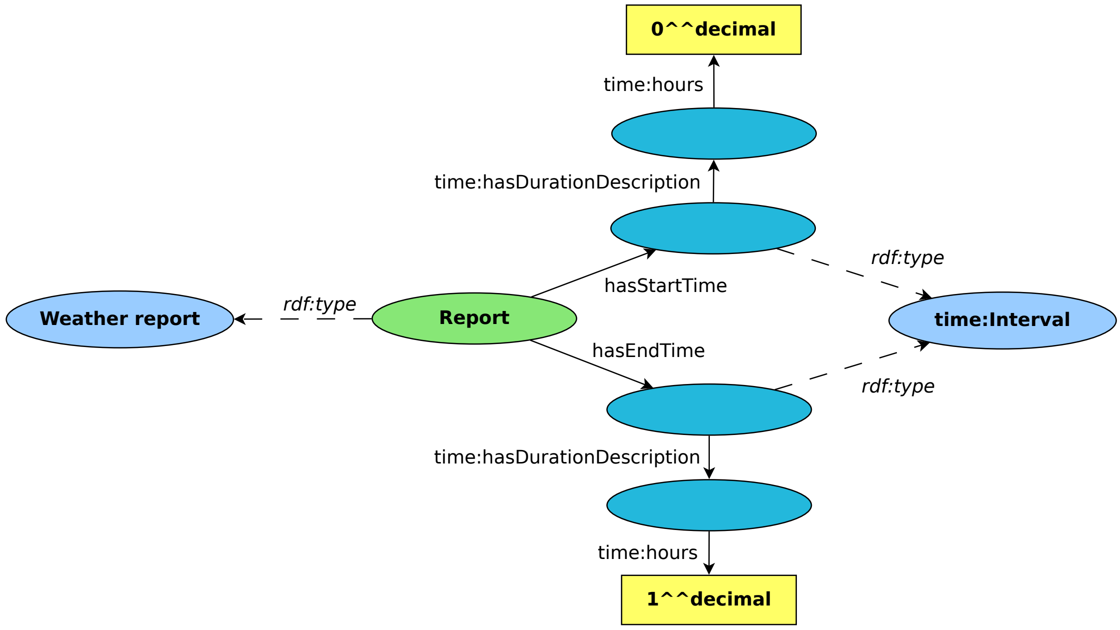

Figure 5.16: An instance of *Current weather report* together with *start time* and *end time*

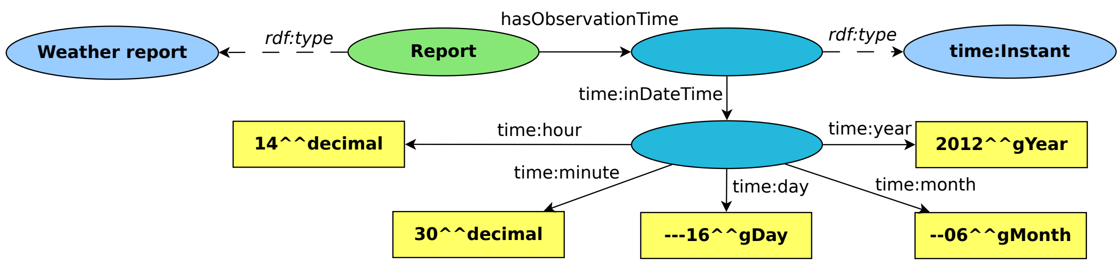

Figure 5.17: An instance of *Weather report* together with its *observation time* (`2012-06-16 14:30`)

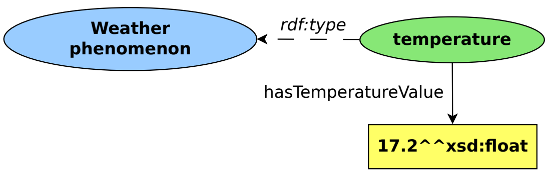

Figure 5.18: An instance of *Temperature* together with the property *has temperature value* representing a temperature of 17.2 °C (without using a unit ontology)

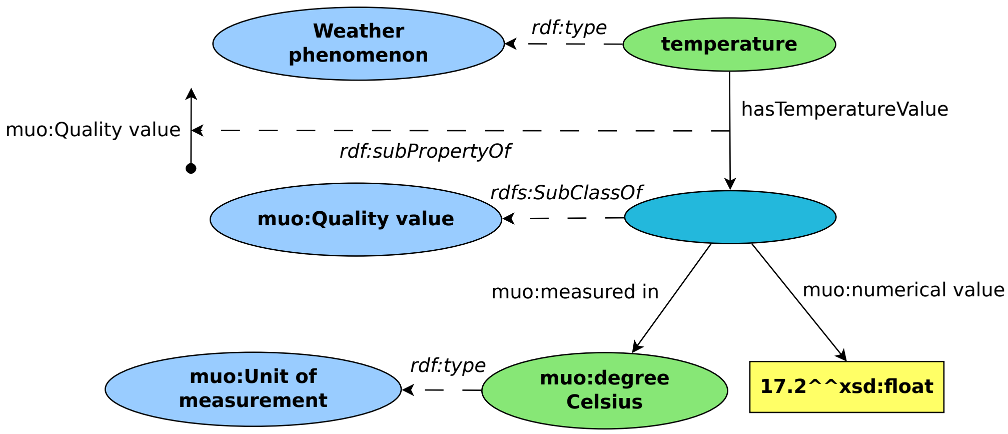

Figure 5.19: An instance of *Temperature* together with the property *has temperature value* representing a temperature of 17.2 °C (using *MUO*)

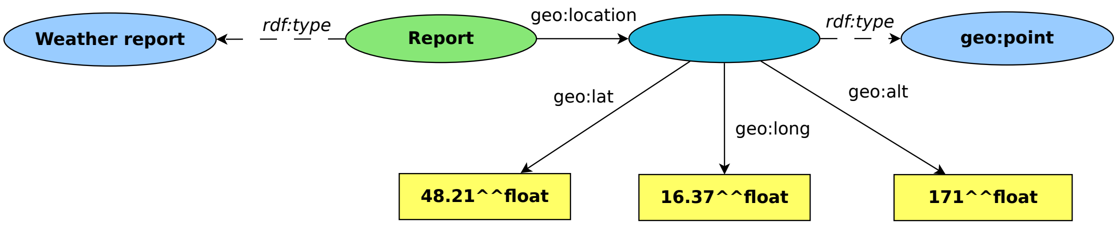

Figure 5.20: An instance of *Weather report* together with its *location* (Vienna, Austria: N 48.21°, E 16.37°, 171 m above *MSL*).

5.7 Evaluation

204-205
2 notes:

After completing the implementation step, this section evaluates *SmartHomeWeather* regarding all non-functional requirements (section 5.7.1) and functional requirements (section 5.7.2) from section 3.1 and the *Ontology Requirements Specification Document* in section 5.2.

5.7.1 Non-functional requirements

There are three non-functional requirements:

- **Naming conventions**: All identifiers in *SmartHomeWeather* follow the naming conventions stated in section 5.1.
- **Documentation**: Due to following the *METHONTOLOGY* approach, the ontology is well documented at every stage of development.
- **Usage of other ontologies**: *SmartHomeWeather* imports the *Basic Geo (WGS84 lat/-long) Vocabulary*, *MUO* and *OWL-Time*; no ontology has been found that satisfies the re-

quirements of *SmartHomeWeather* regarding concepts for weather data, thus *SmartHome-Weather* defines its own concepts.

Thus, all non-functional requirements are met by *SmartHomeWeather*.

5.7.2 Functional requirements

SmartHomeWeather meets its functional requirements if it provides answers to all competency questions. This section evaluates if answers are provided and how answers can be drawn from the ontology.

For the following questions, *SmartHomeWeather* can give straight answers:

- What is the current weather situation?
- What will the weather situation be in one hour, in two hours, ..., in 24 hours?
- What is the current temperature, humidity, wind speed, ...?
- What will be the temperature, humidity, wind speed, ... in one hour, in two hours, ..., in 24 hours?
- Does it rain?

To answer any of these questions, the relevant instance of *Weather report* must be identified. Via the property *has weather state*, an instance of *Weather state* is connected to it; this instance has in turn an arbitrary number of *has weather phenomenon* properties each linking to an instance of *Weather phenomenon*. The information from these instances of *Weather phenomenon* provide the desired answer.

However, there are questions that cannot be answered by *SmartHomeWeather* as writing rules to infer answers would be too complicated in *OWL*:

- Will the weather change? Will the temperature, humidity, ... rise or fall?
- Do we need to irrigate the garden?

Furthermore, there are questions that cannot be answered using an *OWL* ontology due to the *Open World Assumption* [53]. For instance, an *OWL* reasoner cannot determine that some attribute value is the lowest one as it does not know whether there may be lower value it does not know anything about; the reasoner only knows about the presence of individuals and attribute values, but nothing about their absence.

- What will be the minimum temperature, humidity, ... over the next 24 hours? What about maximum values?
- Will it rain in the next hours? Will it rain today?
- Will there be sunshine today?

- Will there be severe weather?
- Will temperature drop/stay below 0 °C?
- When can we open windows and when do we have to keep them shut?
- When do we need sun protection?
- When will it outside be colder than inside the house? When will it be warmer?

However, for each of the above competency questions, whether a direct answer can be given or not, *SmartHomeWeather* can provide all available data; an external program (which is not limited by the *Open World Assumption*) can access this data (e.g. using *SPARQL* queries) and generate an answer.

Hence, the ontology constructed in this chapter complies with its specification in section 5.2.

206
07.08.13
by: Mario J..

CHAPTER 6

The *Weather Importer*

207-208 2 notes: In previous chapters, two topics are discussed that are relevant for this chapter: Chapter 3 digs into the details of weather services that are available via Internet, with the example of the *API* of the *Norwegian Meteorological Institute* (*yr.no*) that is found to best fit the requirements found at the *SmartHomeWeather* project. Chapter 5 describes the design of the *SmartHomeWeather* ontology. Eventually, the ontology needs to be populated with data, i.e. individuals that comprise the current and future state of the weather at the desired location.

For that process, a standalone Java application has been developed. As its main purpose is to import weather data, it was named *Weather Importer*.

At its current state of development, *Weather Importer* obtains data only from *yr.no* in order
209 07.08.13 by: Mario J.. to provide a reference implementation being both simple and functional, but it is designed to allow simple integration of other weather services that are available via Internet as well as data from local weather sensors.

The classes are arranged in two packages, `model` and `main`. The `model` package contains an object-oriented data model for the weather data being processed (see section 6.1). All other classes belong to the `main` package, including the `Main` class providing the `main` method, the classes `TurtleStatement` and `TurtleStore` for output in Turtle *syntax* [48] (see section 6.2.4), the interface `Importer`, and its reference implementation `YrNoImporter` (see section 6.2.1), `WeatherImporterProperties` which encapsulates the *properties file* (see section 6.2), and `WeatherImporterException`, an exception class that is used throughout the application.

All classes belonging to *Weather Importer* include comments suitable for use with the *Javadoc Tool* [212].

6.1 The data model

The core of *Weather Importer* is formed by an object-oriented data model that can be found in the `model` package. This package contains classes that are to be instantiated in order to encapsulate all data that is collected from weather sensors and services. After processing the data in a manner that makes it suitable for use within the *SmartHomeWeather* ontology, individuals and statements are generated and added to the ontology.

The domain model in the package `model` which resembles the structure of the *SmartHome-Weather* ontology is depicted in the *UML* class diagram in figure 6.1; to give an overview, figure 6.2 shows a simplified class diagram that shows only the most important classes `Weather`, `WeatherReport`, `WeatherState`, and `WeatherPhenomenon`.

Other than its name suggests, `OntologyClass` is an interface that is implemented by every class that corresponds to a concept (class) in the ontology. That interface defines a set of methods which are necessary to export an object's data either to individuals and statements for adding them to the ontology using *Apache Jena* [62] or to a representation of the individuals and statements in Turtle *syntax* (see section 6.2 below). The methods defined by `OntologyClass` are:

210
07.08.13
by: Mario J..

- `createIndividuals()` creates individuals and statements holding the data that is stored in the object and adds them to the ontology. The method calls `createIndividuals()` for any objects that are connected to this object, with the exception of instances of `WeatherState` and `WeatherReport` that are linked together via the properties `previousState` and `previousReport`, respectively.

- `getIndividual()` returns the `Individual` object previously created by the method `createIndividuals()` that represents the main ontology individual described by the object. In some classes, due to the structure of the *SmartHomeWeather* ontology and the ontologies being imported, calling `createIndividuals()` will add more than one individual to the ontology, e.g. an `Instant` creates an individual of type *Instant* and one of type *DateTimeDescription*[1]. `getIndividual()` will return the `Individual` object representing the *Instant* individual; the corresponding *DateTimeDescription* object can be obtained by querying the ontology for the statement having the previously returned *Instant* individual as subject and the property *inDateTime* as predicate.

- `getTurtleStatements()` returns an instance of `TurtleStore` that contains a set of `TurtleStatement` objects each representing an *RDF* triple in Turtle *syntax*. The instance being returned contains all data that calling `createIndividuals()` would add to the ontology. See section 6.2.4 for details.

- `getTurtleName()` returns the qualified name of the individual represented by the object that is used in in the `TurtleStatements` returned by calling `getTurtleStatements()`. In case `createIndividuals()` creates more than one individual, the method only yields the name of the individual returned by `getIndividual()`.

- `toString()` returns a textual representation of the object (for debugging purposes).

The classes inside the package `model` are:

[1] *DateTimeDescription* is defined by *OWL-Time* as the concept that represents the timestamp of an *Instant*; the property *inDateTime* links an instance of *DateTimeDescription* to an instance of *Instant*.

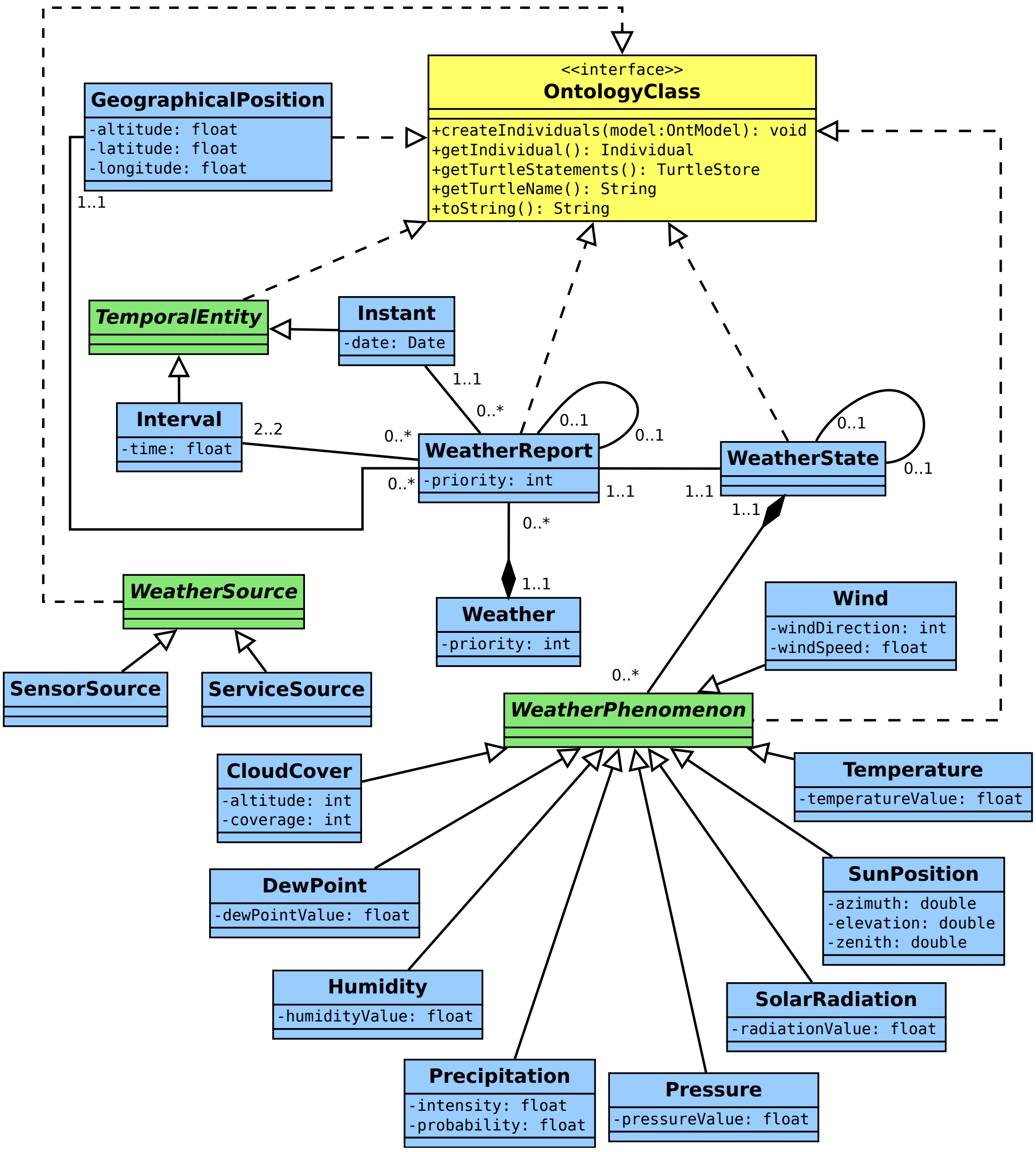

Figure 6.1: The domain model used in *Weather importer*. See figure 6.2 for a simplified diagram that shows only the most important classes.

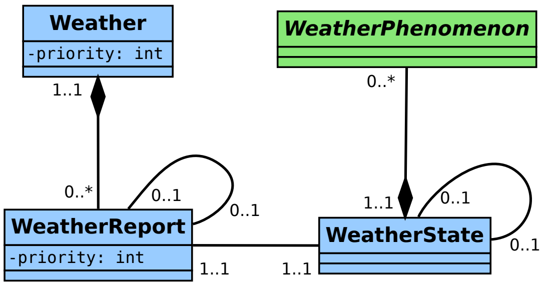

Figure 6.2: The most important classes of the domain model used in *Weather importer*. Refer to figure 6.1 for a diagram showing all classes.

- `GeographicalPosition` resembles the concept *Point* of the *Basic Geo (WGS84 lat/long) Vocabulary* [91] that is imported into the *SmartHomeWeather* ontology.

- `TemporalEntity` corresponds to the concept *Temporal entity* in the *OWL-Time* [95] ontology. The are two sub-classes, `Instant` and `Interval` that resemble the concepts *Instant* and *Interval*, respectively.

- `WeatherPhenomenon` corresponds to the concept *Weather phenomenon* in the *SmartHomeWeather* ontology. As it is an abstract class, only its subclasses `CloudCover`, `DewPoint`, `Humidity`, `Precipitation`, `Pressure`, `SolarRadiation`, `SunPosition`, `Temperature` and `Wind` can be instantiated that each resemble the corresponding concept of the ontology.

- `WeatherReport` corresponds to the concept *Weather report*.

- `WeatherSource` corresponds to the concept *Weather source*. It is an abstract class and has two subclasses `SensorSource` and `ServiceSource` resembling the concepts *Sensor source* and *Service source*, respectively.

- `WeatherState` corresponds to the concept *Weather state*.

- `Weather` has no counterpart in the ontology; it represents a collection of instances of `WeatherReport` which are obtained from sensors and/or services at the same time.

Additionally, there is an enumeration named `WeatherConditions` having values that each correspond to the individuals predefined by the ontology for the concept *Weather condition*.

6.2 The application

211
07.08.13
by: Mario J..

The *Weather Importer* application basically performs three tasks when being launched. It reads the *SmartHomeWeather* ontology in *RDF/XML syntax* [46] from a file, modifies it in some way and and writes the modified ontology into another file, either in *RDF/XML syntax* or in Turtle *syntax* [48]. There are four operation modes that are covered below: `fetch`, `timestamps`, `remove` and `turtle`.

The application depends on the *Apache Jena* framework [62] (successfully tested with 2.10.0). For the unit tests (see section 6.3), *JUnit* [210] (4.11), the *Pellet OWL* 2 reasoner [58] (2.3.0) and *Cobertura* [213] (1.9.4.1) are used. The version numbers given in parentheses give the versions of the most recent releases of the libraries at the time of writing. Newer releases may work, but have not been tested.

Weather Importer comes with a build script for *Apache Ant* [214] that provides target definitions for compiling, running and testing the application:

- The targets `compile` and `compile_test` compile the application and the *JUnit* test cases, respectively. `dist` generates two *JAR* files [215], one containing the application and one for the class that imports weather data from *yr.no*. `clean` removes all files and directories generated by the aforementioned targets and the target `javadoc`; `rebuild` executes `clean`, `compile`, `dist` and `compile_test` consecutively.

- The targets `fetch`, `timestamps`, `remove` and `turtle` launch the application in the respective modes.

- The target `test` runs the *JUnit* test cases; `coverage` generates a coverage report using *Cobertura*, i.e. an overview about which parts of the application's code are covered by the test cases (see section 6.3 for details).

- The target `javadoc` generates documentation from comments in the source code using the *Javadoc Tool* [212].

Various parameters of *Weather Importer* are configurable using a *properties file* [216] which provides the location for which weather data shall be fetched (given by latitude, longitude and altitude), the timestamps relative to the current time in hours for which instances of `WeatherReport` shall be created, names of input and output files and the name of the class that fetches weather data. Additional options required by an implementation of the `Importer` interface may be added.

6.2.1 `fetch` mode

In `fetch` mode, *Weather Importer* reads the *SmartHomeWeather* ontology in *RDF/XML syntax* from a file using the *Apache Jena* framework and fetches weather data for the desired location from a weather service via Internet.

To provide the reference implementation that is found in the class `YrNoImporter`, *Weather Importer* obtains weather data from *yr.no* as described in section 3.5. Any other sources for weather data, regardless whether that sources are weather sensors, Internet weather services or any combination of a set of these, can be utilized by creating a class that implements the interface `Importer`. This interface defines a single method named `fetchWeather()` that returns a `Weather` object containing all weather data obtained from sensors and/or services.

213-214 2 notes: By calling the method `createIndividuals()` of that `Weather` object, the weather data is added *Apache Jena*'s in-memory representation of the ontology. Eventually, the modified ontology is written back to a file in *RDF/XML* syntax.

As most weather services do not provide data for arbitrary points of time, the `Weather` class provides the method `normalizeWeatherReports()`. It transforms the data encapsulated by the `Weather` object in the following ways:

- Each associated `WeatherReport` object that covers a period of more than one hour is replaced by several `WeatherReport` objects, one for each hour. All associated instances of `WeatherReport` and `WeatherPhenomenon` are cloned appropriately.
- If there is more than one `WeatherReport` object covering the same period of time, all data from these objects are merged into one object; the remaining objects will be discarded.
- In case there is no data for a period of time, it is calculated using linear interpolation from data before and after the missing period [217].
- An instance of `SunPosition` is associated to each instance of `WeatherState`. The sun position data is calculated using the *PSA algorithm* [164] (refer to section 3.6 for details); the C++ reference implementation of the *PSA algorithm* [218] was ported to Java.

Additionally, the class `WeatherState` provides the method `mergePhenomena()` which merges all instances of `WeatherPhenomenon` of the same type that are associated to that instance of `WeatherState`. Actual merging of values takes place in the constructors of the subclasses of `WeatherPhenomenon`; all current implementations merge values by calculating the arithmetic mean of all values provided [217].

Both methods provide the developer of an implementation of the interface `Importer` with more flexibility on how to import weather data: There is no need to create a separate instance of `WeatherReport` for every possible period of time; each `WeatherReport` object may cover more than one hour and more than one instance of each subclass of `WeatherPhenomenon` may be associated to each instance of `WeatherState`. The latter eases merging values from several sources (e.g. an Internet weather service and a set of weather sensors).

6.2.2 `timestamps` mode

There are two ways to update weather data in the *SmartHomeWeather* ontology:

- The data can be reobtained using the `fetch` mode into a copy of the ontology that does not contain any weather data. If it does contain any weather data, it can be removed using the `remove` mode (see below).
- 215 07.08.13 by: Mario J.. Alternatively, the timestamps of all instances of the *Weather report* concept in order to make them correspond to the current time.

```
weather:interval0.0          time:hasDurationDescription          weather:hour0.0  .
weather:interval1.0          time:hasDurationDescription          weather:hour1.0  .
weather:interval2.0          time:hasDurationDescription          weather:hour2.0  .
weather:interval3.0          time:hasDurationDescription          weather:hour3.0  .
weather:interval4.0          time:hasDurationDescription          weather:hour3.0  .

weather:weatherReport2       weather:hasStartTime                 weather:interval2.0  ;
                             weather:hasEndTime                   weather:interval3.0  .

weather:weatherReport3       weather:hasStartTime                 weather:interval3.0  ;
                             weather:hasEndTime                   weather:interval4.0  .

weather:weatherReport2       weather:hasObservationTime           weather:instant0  .
weather:weatherReport2       weather:hasObservationTime           weather:instant0  .

weather:instant0             time:inDateTime                      weather:dateTime0  .

weather:dateTime0            a
time:DateTimeDescription  ;
                             time:unitType                        time:unitMinute  ;
                             time:minute                          44  ;
                             time:hour                            12  ;
                             time:day                             "---02"^^xsd:gDay  ;
                             time:month                           "--03"^^xsd:gMonth  ;
                             time:year                            "2013"^^xsd:gYear  .
```

Figure 6.3: Example statements generated by *Weather Importer* running in `fetch` mode.

The latter option is implemented in *Weather Importer* as the `timestamps` mode. That mode is based on the timestamps of each *Weather report* individual being specified by the difference to the current time in hours.

In `timestamp` mode, for each *Weather report* individual stored in the ontology its observation time is retrieved and the difference to the current time in hours is calculated. This difference is then subtracted from both the individual's start time and end time properties; the difference is added to the individual's observation time. Figure 6.3 shows a part of the statements generated by running *Weather Importer* in `fetch` mode; figure 6.4 shows the statements that have been modified by *Weather Importer* running in `timestamps` mode two hours later.

After *Weather Importer* has finished, the *OWL* reasoner must be run using the new data in order to update all knowledge that is produced by the reasoner. E.g., in the example shown in figures 6.3 and 6.4, an instance of *Weather report* that was previously reasoned to be an instance of the concept *Forecast 2 hours weather report* becomes an instance of *Current weather report*, an instance of *Forecast 3 hours weather report* becomes an instance of *Forecast 1 hours weather report* and so on.

6.2.3 `remove` mode

In `remove` mode, the *Weather Importer* takes an ontology in *RDF/XML syntax* from a file using *Apache Jena*. All weather data is removed and the resulting ontology is written back to a file in *RDF/XML syntax*. This file can then be used as input to *Weather Importer*'s `fetch` mode.

weather:interval0.0	time:hasDurationDescription	weather:hour0.0 .
weather:interval1.0	time:hasDurationDescription	weather:hour1.0 .
weather:interval2.0	time:hasDurationDescription	weather:hour2.0 .
weather:interval3.0	time:hasDurationDescription	weather:hour3.0 .
weather:interval4.0	time:hasDurationDescription	weather:hour3.0 .
weather:weatherReport2	weather:hasStartTime	weather:interval0.0 ;
	weather:hasEndTime	weather:interval1.0 .
weather:weatherReport3	weather:hasStartTime	weather:interval1.0 ;
	weather:hasEndTime	weather:interval2.0 .
weather:weatherReport2	weather:hasObservationTime	weather:instant0 .
weather:weatherReport2	weather:hasObservationTime	weather:instant0 .
weather:instant0	time:inDateTime	weather:dateTime0 .
weather:dateTime0	a	
time:DateTimeDescription ;		
	time:unitType	time:unitMinute ;
	time:minute	58 ;
	time:hour	14 ;
	time:day	"−−−02"^^xsd:gDay ;
	time:month	"−−03"^^xsd:gMonth ;
	time:year	"2013"^^xsd:gYear .

Figure 6.4: Example statements modified by *Weather Importer* running in `timestamps` mode about two hours after the running it in `fetch` mode. See figure 6.3 for the statements generated in the initial run; statements modified in `timestamps` mode are highlighted.

6.2.4 `turtle` mode

The `turtle` mode is a mode that was created for debugging reasons; in that mode *Weather Importer* performs the same steps as in `fetch` mode, with the following differences:

- The *SmartHomeWeather* ontology is not read from a file. Hence, the output consists only of the statements generated from the weather data that is imported.
- The *Apache Jena* framework is not used. This enables a developer to distinguish between an error in the usage of *Apache Jena* or an error somewhere else.
- For better readability, Turtle *syntax* is used for output instead of *RDF/XML*.

The `turtle` mode is not necessary for productive use of *Weather Importer*. However, it is kept for providing a demonstrative description of *Weather Importer*'s output and for easing future debugging, if necessary.

Figure 6.5 shows the two classes `TurtleStatement` and `TurtleStore` that provide a data structure for output in Turtle *syntax*. `TurtleStatement` represents a single *RDF* statement in turtle syntax; `TurtleStore` encapsulates a set of `TurtleStatement` objects and provides a method for writing all statements to a file.

Section A.2 in the appendix shows a part of the output generated by *Weather Importer* in `turtle` mode.

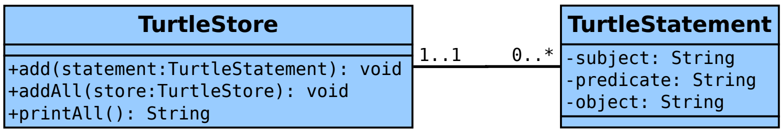

Figure 6.5: Classes used for output in Turtle *syntax*

6.3 Unit tests

Weather Importer incorporates a set of *JUnit* [210] tests that covers reasoning in the *SmartHome-Weather* ontology and the application itself. For testing correct reasoning, the *Apache Jena* framework and the *Pellet* reasoner are used. Additionally, *Cobertura* extends the *JUnit* framework by the generation of a coverage report that lists in detail how often the lines in all Java class files are executed during the unit tests. Using *Cobertura*, it is possible to determine if every line of code that is intended to be tested is actually tested. According to the analysis generated by *Cobertura*, the coverage is 100 % for all classes that are target of unit tests.

The following test categories are implemented:

- **Category 1:** Tests for *OWL* reasoning concerning single individuals of the ontology; e.g. an instance of *Weather phenomenon* that has a *has temperature value* property must be reasoned to be an instance of *temperature*.
- **Category 2:** Tests that involve reasoning for instances of several concepts; e.g. correct reasoning of *Calm weather*.
- **Category 3:** Tests for the import of weather data from *yr.no*.
- **Category 4:** Tests for the output in Turtle *syntax*.

The class `Main` in the package `main` remains not being covered by *JUnit* tests; its purpose is to only read command input and to instantiate the appropriate classes which are covered by *JUnit* anyway.

All test cases of *category 1* and *category 2* share the same approach:

1. Using *Apache Jena*, the *SmartHomeWeather* ontology is read from its *RDF/XML* representation from disk and an in-memory representation is created.
2. One or more test individuals together with their properties are added to the in-memory ontology. E.g. when testing the correct functionality of the sub-concept *Room temperature* of *Temperature*, an instance of *Weather phenomenon* is generated together with a property of type *has temperature value* that assigns the value of 20 °C to the *Weather phenomenon* using the *MUO* ontology.
3. The *Pellet* reasoner is invoked to obtain all statements that can be inferred from the currently available knowledge base.

4. The set of all statements that include the previously generated instance(s) is compared to a predefined set of expected statements. For the test case to be successful, the two sets must match.

   In the above example of testing *Room temperature*, the created instance of *Weather phenomenon* must be inferred to be an instance of *Weather phenomenon*, *Temperature*, and *Room temperature*. Additionally, it must not be an instance of *Frost*, *Cold*, *Below room temperature*, *Above room temperature* or *Heat*.

5. The in-memory representation of *SmartHomeWeather* is destroyed.

6. The steps 1 to 5 are repeated to perform another test. When testing *Room temperature*, another temperature value could be chosen or another sub-concept of *Temperature* could be selected. Once the whole range that is intended to be tested, another concept will be tested; *Temperature* is tested in the range from $-100\,°\mathrm{C}$ to $100\,°\mathrm{C}$, with a resolution of $0.1\,°\mathrm{C}$ between $0\,°\mathrm{C}$ and $30\,°\mathrm{C}$ and $0.5\,°\mathrm{C}$ otherwise.

216
07.08.13
by: Mario J..

During a full test run, the above steps are executed 4620 times for *category 1* test cases and 2250 times for *category 2* test cases.

A test of *category 3* is implemented using a pre-defined snippet containing an *XML* document as it is returned by the *API* or *yr.no*. This snippet is fed into the *Weather Importer* to generate an object-oriented data model. Then it is verified whether this data model contains exactly the data from the snippet. Some of the snippets are invalid, i.e. they contain *XML* code as *yr.no* is not allowed to return. In that case, the *Weather Importer* is expected to fail with an appropriate error message. The total number of different snippets used for *category 3 JUnit* test cases is 31.

217
07.08.13
by: Mario J..

During a test of *category 4*, an object-oriented weather data model is generated which is then exported to *Turtle* syntax. This output is then compared to the expected output. There are 12 such test cases.

CHAPTER 7

Conclusion

218
07.08.13
by: Mario J..

7.1 Summary

219
07.08.13
by: Mario J..

The previous chapters present a weather data model based on an *OWL* ontology, a Java application for importing data into that model, and all prerequisites leading to these two outcomes.

220
07.08.13
by: Mario J..

After chapter 1 gives an introduction into the motivation behind the thesis and describes the problem statement, the intended goal and the methodological approach, chapter 2 digs into the foundations the thesis builds upon: Ontologies, ontology related technologies like *RDF*, *RDFS*, or *OWL*, *ThinkHome*, already existing ontologies for weather data, and ontologies that cover information related to the domain covered by *SmartHomeWeather*. Chapter 3 then focuses on weather data that is available from both locally installed weather sensors as well as from weather services accessible via Internet. This chapter then identifies a set of weather phenomena which *SmartHomeWeather* shall cover and determines further details about the domain of weather data as used in the present context. Of the weather services being discussed, *yr.no* is selected for providing a reference implementation for the import of weather data into *SmartHomeWeather* in a later step. Chapter 4 sheds light on five different approaches towards the development of new ontologies from scratch. The approaches are compared to each other and one of them, *METHONTOLOGY*, is identified as the one that fits the requirements for designing *SmartHomeWeather* best.

Eventually, chapter 5 applies *METHONTOLOGY* to *SmartHomeWeather* and describes every step in a detailed manner. As the data model itself does not contain any weather data, in chapter 6 the *Weather Importer*, a Java application, is developed which accesses *yr.no* to retrieve weather data for a certain location, transforms the data being obtained and adds them to the data model of *SmartHomeWeather*. Furthermore, the *Weather Importer* includes a comprehensive set of *JUnit* test cases which ensure that *SmartHomeWeather* and the *Weather Importer* work as expected.

7.2 Outlook

The current version of *SmartHomeWeather* comes with a few problems that require future work in order to resolve them. Not all of these problems lie in the scope of *SmartHomeWeather*. The main problems arising are:

- There are a few situations that expose bugs in *Protégé* and the *Pellet* reasoner. While it is possible to develop ontologies using these technologies which cover a certain domain exactly in the way required by a specification, it is hardly avoidable to run into bugs that manifest themselves in the form of incomprehensible error messages. In that case, the ontology needs to be modified slightly to work around these bugs. Unfortunately, during the development of *SmartHomeWeather*, it has not been possible to track these bugs down in order to find their reason and to fix them. Future work that resolves these bugs may ease the work with *Protégé*, *Pellet*, and *SmartHomeWeather*.

- At the time of writing, *OWL-Time* [95] (see section 2.3.2) has not reached the state of being a W3C *recommendation* [51]; since it was first published more than six years ago, it is a *working draft*. Although it can be assumed that the core concepts and relations defined by *OWL-Time* will not change regarding their syntax and semantics, using technologies that are not yet published as being in a stable state always leaves a stale taste.

- Similar problems arise from the use of the *Basic Geo (WGS84 lat/long) Vocabulary* [91]. This technology has not even been submitted to the W3C *recommendation track* for standardisation. Furthermore, no work on the WGS84 *vocabulary* itself has been done since 2006. Further work by the W3C *Geospatial Incubator Group* did not lead to any standards (see 2.3.1).

Furthermore, *SmartHomeWeather* suffers performance issues regarding the time required for reasoning. In test runs that were conducted after development, complete reasoning in *Protégé* using the *Pellet reasoner* of the "empty" ontology (not containing any individuals denoting weather data) took between 15 and 30 seconds and between 45 and 60 seconds for the ontology that contains weather data imported using the *Weather Importer* from chapter 6 (on the PC used for development which is equipped with a *Intel Q6600 CPU* [219] running *Ubuntu Linux* [220]).

One reason for this performance issues is the use of the *MUO* ontology which increases the reasoning time by about 30% (see section 5.6.1). Abandonment of this ontology would speed up refactoring, though this would introduce problems regarding literal values without a unit. There are other ontologies which may be used instead of the *MUO* ontology (see section 2.3.3), such as the *OM* ontology. However, as the *OM* ontology adds about the same level of complexity to the ontology, it may increase the reasoning time of the smart home's knowledge base to the same extent as *MUO*. It is questionable whether there is a unit ontology that allows faster reasoning than *MUO*.

Further performance optimisations may be possible by modifying the internal structure of *SmartHomeWeather* without changing name and semantics of externally accessed concepts. E.g. concepts such as *Weather report*, *Weather state*, and *Weather phenomenon* together with their respective sub-concepts remain part of the ontology while the definitions of these concepts are

227 07.08.13 by: Mario J..

modified to allow faster reasoning. However, it is unknown to what degree performance gains are possible using this approach.

228 07.08.13 by: Mario J..

229 07.08.13 by: Mario J..

Nevertheless, in its current state, *SmartHomeWeather* represents an ontology that fully complies with its specification of covering current and future weather data as far as possible. Although being envisioned in the context of *ThinkHome*, it qualifies to be used in any ontology-based smart home system as well. Eventually, the employment of *SmartHomeWeather* in environments other than smart homes is also imaginable, whereever current and future weather data is used.

APPENDIX A

Tables and listings

This appendix contains tables and listings that are referenced from other chapters.

A.1 Conceptualisation tables for *SmartHomeWeather*

In order to keep the documentation of *SmartHomeWeather* clear, a set of tables is omitted from section 5.4. This section contains these tables in case they are needed for reference.

The tables in this section are:

- *Concept dictionaries* for *Weather condition*, *Weather report*, and *Weather source* (figure A.1), and for *Weather phenomenon* and *Weather state* (figure A.2); see section 5.4.4 for details about *concept dictionaries*.
- The *Binary relations table* in figure A.3; see section 5.4.5 for details.
- The *Instance attributes table* in figure A.4; see section 5.4.6 for details.
- The *Class attributes table* in figure A.5, figure A.6, figure A.7, and figure A.7; see section 5.4.7 for details.
- The *Instances table* in figure A.9; see section 5.4.8 for details.

Name	Instances	Relations
Weather condition	*cloud*, *fog*, *partly cloudy*, *mostly cloudy*, *rain*, *sleet*, *snow*, *sun*, *thunder*	*has condition*
Weather state		*has condition*, *belongs to weather report*, *has weather state*, *belongs to state*, *has weather phenomenon*
Weather source		*is source of*, *has source*
Sensor source		*is source of*, *has source*
Service source		*is source of*, *has source*

Table A.1: Concept dictionary for *Weather condition*, *Weather report*, *Weather state*, and *Weather source*

Name	**Instance attributes**	**Relations**
Atmospheric pressure	*has pressure value*	*belongs to state*, *has weather phenomenon*
Dew point	*has dew point value*	*belongs to state*, *has weather phenomenon*
Humidity	*has humidity value*	*belongs to state*, *has weather phenomenon*
Precipitation	*has precipitation intensity*, *has precipitation probability*	*belongs to state*, *has weather phenomenon*
Sun position	*has sun elevation angle*, *has sun direction*	*belongs to state*, *has weather phenomenon*
Solar radiation	*has solar radiation value*	*belongs to state*, *has weather phenomenon*
Temperature	*has temperature value*	*belongs to state*, *has weather phenomenon*
Weather phenomenon	-	*belongs to state*, *has weather phenomenon*
Wind	*has wind speed*, *has wind direction*	*belongs to state*, *has weather phenomenon*
weather report	*has priority*	*has source*, *is source of*, *has weather state*, *belongs to weather report location*, *has start time*, *has end time*, *has observation time*
weather report	*has priority*	*has source*, *is source of*, *has weather state*, *belongs to weather report location*, *has start time*, *has end time*, *has observation time*

Table A.2: Concept dictionary for *weather phenomenon* and *weather report*

Name	Source concept	Target concept	Maximum source cardinality	Inverse relation
belongs to state	*Weather phenomenon*	*Weather state*	1	*has weather phenomenon*
belongs to weather report	*Weather state*	*Weather report*	1	*has weather state*
has condition	*Weather state*	*Weather condition*	N	-
has end time	*Weather report*	*Interval*	1	-
has observation time	*Weather report*	*Instant*	1	-
has next weather state	*Weather report*	*Weather report*	1	*has previous weather state*
has previous weather state	*Weather report*	*Weather report*	1	*has next weather state*
has source	*Weather report*	*Weather source*	1	*is source of*
has start time	*Weather report*	*Interval*	1	-
has weather phenomenon	*Weather state*	*Weather phenomenon*	N	*belongs to state*
has weather state	*Weather report*	*Weather state*	1	*belongs to weather report*
is source of	*Weather source*	*Weather report*	N	*has source*
location	*Weather report*	*Point*	1	-

Table A.3: Binary relations table

Attribute name	**Concept name**	**Value type**	**Value range**	**Unit**	**Cardinality** (min, max)
alt	*Location*	*xsd:decimal*	*any values allowed*	m	$(1, 1)$
has cloud altitude	*Cloud cover*	*xsd:decimal*	*any values allowed*	m	$(1, 1)$
has cloud cover	*Cloud cover*	*xsd:integer*	$[0, 9]$	*Okta*	$(1, 1)$
has dew point value	*Dew point*	*xsd:decimal*	*any values allowed*	°C	$(1, 1)$
has humidity value	*Humidity*	*xsd:decimal*	$[0, 1]$	-	$(1, 1)$
has precipitation intensity	*Precipitation*	*xsd:decimal*	$[0, \infty)$	mm/h	$(1, 1)$
has precipitation probability	*Precipitation*	*xsd:decimal*	$[0, 1]$	-	$(1, 1)$
has pressure value	*Atmospheric pressure*	*xsd:decimal*	$[0, \infty)$	hPa	$(1, 1)$
has solar radiation value	*Solar radiation*	*time:decimal*	$[0, \infty)$	W/m^2	$(1, 1)$
has sun direction	*Sun position*	*xsd:decimal*	$[0, 360)$	° (degrees)	$(1, 1)$
has sun elevation angle	*Sun position*	*xsd:decimal*	$[-90, 90]$	° (degrees)	$(1, 1)$
has temperature value	*Temperature*	*xsd:decimal*	*any values allowed*	°C	$(1, 1)$
has wind direction	*Wind*	*xsd:decimal*	$[0, 360)$	° (degrees)	$(1, 1)$
has wind speed	*Wind*	*xsd:decimal*	$[0, \infty)$	m/s	$(1, 1)$
lat	*Location*	*xsd:decimal*	$[-90, 90]$	° (degrees)	$(1, 1)$
long	*Location*	*xsd:decimal*	$[-180, 180]$	° (degrees)	$(1, 1)$

Table A.4: Instance attributes table

Super-concept	**Sub-concept**	**attribute name**	**attribute value(s)**
Atmospheric pressure	*Very low pressure*	*has pressure value*	< 998
Atmospheric pressure	*Low pressure*	*has pressure value*	$[998, 1008)$
Atmospheric pressure	*Average pressure*	*has pressure value*	$[1008, 1018)$
Atmospheric pressure	*High pressure*	*has pressure value*	$[1018, 1028)$
Atmospheric pressure	*Very high pressure*	*has pressure value*	≥ 1028
Cloud cover	*Clear sky*	*has cloud cover*	0
Cloud cover	*Partly cloudy*	*has cloud cover*	1, 2, 3, 4
Cloud cover	*Mostly cloudy*	*has cloud cover*	5, 6, 7
Cloud cover	*Overcast*	*has cloud cover*	8
Cloud cover	*Unknown cloud cover*	*has cloud cover*	9
Humidity	*Very dry*	*has humidity value*	< 0.3
Humidity	*Dry*	*has humidity value*	$[0.3, 0.4)$
Humidity	*Normal humidity*	*has humidity value*	$[0.4, 0.7]$
Humidity	*Moist*	*has humidity value*	$(0.7, 0.8]$
Humidity	*Very moist*	*has humidity value*	> 0.8
Precipitation	*No rain*	*has precipitation intensity* *has precipitation probability*	0 0
Precipitation	*Light rain*	*has precipitation intensity* *has precipitation probability*	$(0, 5]$ $(0, 1]$
Precipitation	*Medium rain*	*has precipitation intensity* *has precipitation probability*	$(5, 20]$ $(0, 1]$
Precipitation	*Heavy rain*	*has precipitation intensity* *has precipitation probability*	$(20, 50]$ $(0, 1]$

Table A.5: Class attributes table (1)

Super-concept	**Sub-concept**	**attribute name**	**attribute value(s)**
Precipitation	*Extremely heavy rain*	*has precipitation intensity* *has precipitation probability*	$(50, 100]$ $(0, 1]$
Precipitation	*Tropical storm rain*	*has precipitation intensity* *has precipitation probability*	> 100 $(0, 1]$
Solar radiation	*No radiation*	*has solar radiation value*	0
Solar radiation	*Low radiation*	*has solar radiation value*	$(0, 250)$
Solar radiation	*Medium radiation*	*has solar radiation value*	$[250, 500)$
Solar radiation	*High radiation*	*has solar radiation value*	$[500, 750)$
Solar radiation	*Very high radiation*	*has solar radiation value*	≥ 750
Sun position	*sun from north*	*has sun direction*	$[0, 45] \cup (315, 360)$
Sun position	*sun from east*	*has sun direction*	$(45, 135]$
Sun position	*sun from south*	*has sun direction*	$(135, 225]$
Sun position	*sun from west*	*has sun direction*	$(225, 315]$
Sun position	*day*	*has sun elevation angle*	$[0, 90]$
Sun position	*solar twilight*	*has sun elevation angle*	$[0, 6)$
Sun position	*sun below horizon*	*has sun elevation angle*	$[-90, 0)$
Sun position	*twilight*	*has sun elevation angle*	$[-18, 0)$
Sun position	*civil twilight*	*has sun elevation angle*	$[-6, 0)$
Sun position	*nautical twilight*	*has sun elevation angle*	$[-12, -6)$
Sun position	*astronomical twilight*	*has sun elevation angle*	$[-18, -12)$
Sun position	*night*	*has sun elevation angle*	$[-90, -18)$

Table A.6: Class attributes table (2)

Super-concept	Sub-concept	attribute name	attribute value(s)
Temperature	*frost*	*has temperature value*	< 0
Temperature	*cold*	*has temperature value*	$[0, 10)$
Temperature	*below room temperature*	*has temperature value*	$[10, 20)$
Temperature	*room temperature*	*has temperature value*	$[20, 25]$
Temperature	*above room temperature*	*has temperature value*	$(25, 30]$
Temperature	*heat*	*has temperature value*	> 30
Wind	*directional wind*	*has wind direction*	$[0, 360)$
Wind	*north wind*	*has wind direction*	$[0, 45) \cup [315, 360)$
Wind	*east wind*	*has wind direction*	$[45, 135)$
Wind	*south wind*	*has wind direction*	$[135, 225)$
Wind	*wind wind*	*has wind direction*	$[225, 315)$
Wind	*calm*	*has wind speed*	$[0, 1)$
Wind	*light wind*	*has wind speed*	$[1, 10)$
Wind	*strong wind*	*has wind speed*	$[10, 20)$
Wind	*storm*	*has wind speed*	≥ 20
Wind	*hurricane*	*has wind speed*	≥ 32
weather report	*Short range weather report*	*has start time*	$(0, 3]$
weather report	*Medium range weather report*	*has start time*	$(3, 12)$
weather report	*Long range weather report*	*has start time*	≥ 12
weather report	*Current weather report*	*has start time*	0
weather report	*Forecast weather report*	*has start time*	> 0
weather report	*Forecast 1 hour weather report*	*has start time*	1
weather report	*Forecast 2 hours weather report*	*has start time*	2
weather report	*Forecast 3 hours weather report*	*has start time*	3

Table A.7: Class attributes table (3)

Super-concept	**Sub-concept**	**attribute name**	**attribute value(s)**
weather report	*Forecast 6 hours weather report*	*has start time*	6
weather report	*Forecast 9 hours weather report*	*has start time*	9
weather report	*Forecast 12 hours weather report*	*has start time*	12
weather report	*Forecast 15 hours weather report*	*has start time*	15
weather report	*Forecast 18 hours weather report*	*has start time*	18
weather report	*Forecast 21 hours weather report*	*has start time*	21
weather report	*Forecast 24 hours weather report*	*has start time*	24
weather report	*Weather report from sensor*	*has source*	*any instance of* Sensor source
weather report	*Weather report from service*	*has source*	*any instance of* Service source
current weather report	*Current weather report from sensor*	*has source*	*any instance of* Sensor source
current weather report	*Current weather report from service*	*has source*	*any instance of* Service source

Table A.8: Class attributes table (4)

Instance name	**Concept name**
Cloud	*Weather condition*
Fog	*Weather condition*
Partly cloudy	*Weather condition*
Mostly cloudy	*Weather condition*
Rain	*Weather condition*
Sleet	*Weather condition*
Snow	*Weather condition*
Sun	*Weather condition*
Thunder	*Weather condition*

Table A.9: Instances table

A.2 Output of *Weather Importer* in *Turtle syntax*

This is a part of the output generated by *Weather Importer* run in `turtle` mode on April 4, 2013 at 14:56. Only statements about the first two individuals of type *Weather report* and the individuals that are connected to them via object properties (excluding *has next weather state*) are included. See section 6.2.4 for details.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix weather: <http://www.semanticweb.org/ontologies/2011/9/ThinkHomeWeather.owl#> .
@prefix time: <http://www.w3.org/2006/time#> .
@prefix wgs: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix muo: <http://purl.oclc.org/NET/muo/muo#> .

weather:weatherReport0      a                               weather:WeatherReport ;
                            weather:hasPriority             421 .

weather:yr_no               a                               weather:ServiceSource .

weather:weatherReport0      weather:hasSource               weather:yr_no .

weather:interval0.0         a                               time:Interval .

weather:hour0.0             a                               weather:Hour ;
                            time:hours                      "0"^^xsd:decimal .

weather:interval0.0         time:hasDurationDescription     weather:hour0.0 .

weather:interval1.0         a                               time:Interval .

weather:hour1.0             a                               weather:Hour ;
                            time:hours                      "1"^^xsd:decimal .

weather:interval1.0         time:hasDurationDescription     weather:hour1.0 .

weather:weatherReport0      weather:hasStartTime            weather:interval0.0 ;
                            weather:hasEndTime              weather:interval1.0 .

weather:point0              a                               wgs:Point ;
                            wgs:lat                         "48.21"^^xsd:float ;
                            wgs:lon                         "16.37"^^xsd:float ;
                            wgs:alt                         "171.0"^^xsd:float .

weather:weatherReport0      wgs:location                    weather:point0 .

weather:instant0            a                               time:Instant .

weather:dateTime0           a                               time:DateTimeDescription ;
                            time:unitType                   time:unitMinute ;
                            time:minute                     56 ;
                            time:hour                       14 ;
                            time:day                        "---04"^^xsd:gDay ;
                            time:month                      "--04"^^xsd:gMonth ;
                            time:year                       "2013"^^xsd:gYear .

weather:instant0            time:inDateTime                 weather:dateTime0 .

weather:weatherReport0      weather:hasObservationTime      weather:instant0 .

weather:weatherState0       a                               weather:WeatherState .
```

```
_:blank1                    muo:numericalValue                        "0.0"^^xsd:float ;
                            muo:measuredIn                            weather:percent .
_:blank2                    muo:numericalValue                        "0.0"^^xsd:float ;
                            muo:measuredIn                            muo:millimetresPerHour .
weather:precipitation0.0    a                                         weather:WeatherPhenomenon ;
                            weather:hasPrecipitationProbability       _:blank1 ;
                            weather:hasPrecipitationIntensity         _:blank2 ;
                            weather:belongsToWeatherState             weather:weatherState0 .

_:blank3                    muo:numericalValue                        0 ;
                            muo:measuredIn                            weather:okta .
_:blank4                    muo:numericalValue                        5000 ;
                            muo:measuredIn                            muo:meter .
weather:cloudCover0.0       a                                         weather:WeatherPhenomenon ;
                            weather:hasCloudCover                     _:blank3 ;
                            weather:hasCloudAltitude                  _:blank4 ;
                            weather:belongsToWeatherState             weather:weatherState0 .

_:blank5                    muo:numericalValue                        "7.6"^^xsd:float ;
                            muo:measuredIn                            muo:degrees-Celsius .
weather:temperature0        a                                         weather:WeatherPhenomenon ;
                            weather:hasTemperatureValue               _:blank5 ;
                            weather:belongsToWeatherState             weather:weatherState0 .

_:blank6                    muo:numericalValue                        "0.58"^^xsd:float ;
                            muo:measuredIn                            weather:percent .
weather:humidity0           a                                         weather:WeatherPhenomenon ;
                            weather:hasHumidityValue                  _:blank6 ;
                            weather:belongsToWeatherState             weather:weatherState0 .

_:blank7                    muo:numericalValue                        "-0.8"^^xsd:float ;
                            muo:measuredIn                            muo:degrees-Celsius .
weather:dewPoint0           a                                         weather:WeatherPhenomenon ;
                            weather:hasDewPointValue                  _:blank7 ;
                            weather:belongsToWeatherState             weather:weatherState0 .

_:blank8                    muo:numericalValue                        "1010.0"^^xsd:float ;
                            muo:measuredIn                            weather:hectopascal .
weather:pressure0           a                                         weather:WeatherPhenomenon ;
                            weather:hasPressureValue                  _:blank8 ;
                            weather:belongsToWeatherState             weather:weatherState0 .

_:blank9                    muo:numericalValue                        "1.5"^^xsd:float ;
                            muo:measuredIn                            weather:metresPerSecond .
_:blank10                   muo:numericalValue                        47 ;
                            muo:measuredIn                            muo:degree .
weather:wind0               a                                         weather:WeatherPhenomenon ;
                            weather:hasWindSpeed                      _:blank9 ;
                            weather:hasWindDirection                  _:blank10 ;
                            weather:belongsToWeatherState             weather:weatherState0 .

_:blank11                   muo:numericalValue                        220 ;
                            muo:measuredIn                            muo:degree .
_:blank12                   muo:numericalValue                        "40.76"^^xsd:float ;
                            muo:measuredIn                            muo:degree .
weather:sunPosition0.0      a                                         weather:WeatherPhenomenon ;
                            weather:hasSunDirection                   _:blank11 ;
                            weather:hasSunElevationAngle              _:blank12 ;
                            weather:belongsToWeatherState             weather:weatherState0 .

weather:weatherState0       weather:hasCondition                      weather:PartlyCloud .
```

```
weather:weatherReport1      a                                     weather:WeatherReport ;
                            weather:hasPriority                   421 ;
                            weather:hasSource                     weather:yr_no .

weather:interval2.0         a                                     time:Interval .

weather:hour2.0             a                                     weather:Hour ;
                            time:hours                            "2"^^xsd:decimal .

weather:interval2.0         time:hasDurationDescription           weather:hour2.0 .

weather:weatherReport1      weather:hasStartTime                  weather:interval1.0 ;
                            weather:hasEndTime                    weather:interval2.0 ;
                            wgs:location                          weather:point0 ;
                            weather:hasObservationTime            weather:instant0 .

weather:weatherState1       a                                     weather:WeatherState .

_:blank13                   muo:numericalValue                    "0.0"^^xsd:float ;
                            muo:measuredIn                        weather:percent .
_:blank14                   muo:numericalValue                    "0.0"^^xsd:float ;
                            muo:measuredIn                        muo:millimetresPerHour .
weather:precipitation1.0    a                                     weather:WeatherPhenomenon ;
                            weather:hasPrecipitationProbability   _:blank13 ;
                            weather:hasPrecipitationIntensity     _:blank14 ;
                            weather:belongsToWeatherState         weather:weatherState1 .

_:blank15                   muo:numericalValue                    0 ;
                            muo:measuredIn                        weather:okta .
_:blank16                   muo:numericalValue                    5000 ;
                            muo:measuredIn                        muo:meter .
weather:cloudCover1.0       a                                     weather:WeatherPhenomenon ;
                            weather:hasCloudCover                 _:blank15 ;
                            weather:hasCloudAltitude              _:blank16 ;
                            weather:belongsToWeatherState         weather:weatherState1 .

_:blank17                   muo:numericalValue                    "7.6"^^xsd:float ;
                            muo:measuredIn                        muo:degrees-Celsius .
weather:temperature1        a                                     weather:WeatherPhenomenon ;
                            weather:hasTemperatureValue           _:blank17 ;
                            weather:belongsToWeatherState         weather:weatherState1 .

_:blank18                   muo:numericalValue                    "0.58"^^xsd:float ;
                            muo:measuredIn                        weather:percent .
weather:humidity1           a                                     weather:WeatherPhenomenon ;
                            weather:hasHumidityValue              _:blank18 ;
                            weather:belongsToWeatherState         weather:weatherState1 .

_:blank19                   muo:numericalValue                    "-0.8"^^xsd:float ;
                            muo:measuredIn                        muo:degrees-Celsius .
weather:dewPoint1           a                                     weather:WeatherPhenomenon ;
                            weather:hasDewPointValue              _:blank19 ;
                            weather:belongsToWeatherState         weather:weatherState1 .

_:blank20                   muo:numericalValue                    "1010.0"^^xsd:float ;
                            muo:measuredIn                        weather:hectopascal .
weather:pressure1           a                                     weather:WeatherPhenomenon ;
                            weather:hasPressureValue              _:blank20 ;
                            weather:belongsToWeatherState         weather:weatherState1 .

_:blank21                   muo:numericalValue                    "1.5"^^xsd:float ;
```

```
                        muo:measuredIn                 weather:metresPerSecond .
_:blank22               muo:numericalValue             47 ;
                        muo:measuredIn                 muo:degree .
weather:wind1           a                              weather:WeatherPhenomenon ;
                        weather:hasWindSpeed           _:blank21 ;
                        weather:hasWindDirection       _:blank22 ;
                        weather:belongsToWeatherState  weather:weatherState1 .

_:blank23               muo:numericalValue             236 ;
                        muo:measuredIn                 muo:degree .
_:blank24               muo:numericalValue             "33.42"^^xsd:float ;
                        muo:measuredIn                 muo:degree .
weather:sunPosition1.0  a                              weather:WeatherPhenomenon ;
                        weather:hasSunDirection        _:blank23 ;
                        weather:hasSunElevationAngle   _:blank24 ;
                        weather:belongsToWeatherState  weather:weatherState1 .

weather:weatherState1   weather:hasCondition           weather:PartlyCloud .
```

APPENDIX B

Glossary

A | B | C | D | E | F | H | I | L | M | N | O | P | R | S | T | U | V | W

A

Above room temperature

Sub-concept of *Temperature* representing a temperature of more than 25 and less than or equal to 30 °C. 68, 69, 73, 106, *see* Temperature

Airing weather

Sub-concept of *Weather state* representing a *Weather state* that is an instance of *Fair weather* and *Pleasant temperature weather* at the same time.. 69, *see* Weather state, Fair weather *and* Pleasant temperature weather

Alt

Property defined by the *Basic Geo (WGS84 lat/long) Vocabulary* [91] specifying the height above *MSL* (*mean sea level*) of the *location* the *weather report* is valid for, in metres. 19, 103, *see* Weather report *and* Point

Astronomical twilight

Sub-concept of *sun position* that represents the sun being more than 12 degrees and no more than 18 degrees below horizon. 69, 73, 105, *see* Sun position *and* Twilight

Atmospheric pressure

Sub-concept of *Weather phenomenon* representing atmospheric pressure. The value is specified using the property *has pressure value*. Sub-concepts are *Very low pressure*, *Low pressure*, *Average pressure*, *High pressure*, and *Very high pressure*. 68, 70, 71, 78, 101, 103, 104, 134, *see* Weather phenomenon, Has pressure value, Very low pressure, Low pressure, Average pressure, High pressure *and* Very high pressure

Average pressure

Sub-concept of *Atmospheric pressure* representing a pressure of at least 1008 and less than 1018 hPa. 68, 70, 78, 104, *see* Atmospheric pressure

B

Belongs to state

Object property that links instances of *Weather state* and *Weather phenomenon*; inverse property of *has weather phenomenon*. 69, 100–102, *see* Weather state, Weather phenomenon *and* Has weather phenomenon

Belongs to weather report

Object property that links instances of *Weather report* and *Weather state*; inverse property of *has weather state*. 69, 100–102, *see* Weather report, Weather state *and* Has weather state

Below room temperature

Sub-concept of *Temperature* representing a temperature of more than or equal to 10 and less than 20 °C. 68, 69, 73, 106, *see* Temperature

C

Calm

Sub-concept of *Wind* representing wind with a speed below 1 m/s. 69, 75, 106, *see* Wind

Calm weather

Sub-concept of *Weather state* representing a *Weather state* that is linked to an instance of *Calm* and an instance of *Light wind* via the property *has weather phenomenon*. 69, 93, *see* Weather state, Has weather phenomenon, Calm *and* Light wind

Civil twilight

Sub-concept of *sun position* that represents the sun being below horizon and at most 6 degrees below horizon. 69, 73, 105, *see* Sun position *and* Twilight

Clear sky

Sub-concept of *Cloud cover* representing cloud coverage that is reported to be 0 *okta*. 69, 70, 104, *see* Okta *and* Cloud cover

Clear weather

Sub-concept of *Weather state* representing a *Weather state* that is linked to an instance of *Clear sky* or *Partly cloudy* via the property *has weather phenomenon*. 69, *see* Weather state, Has weather phenomenon, Clear sky *and* Partly cloudy

Cloud cover

Sub-concept of *Weather phenomenon* describing the current cloud coverage in *okta* (integer numbers from 0 to 9) which is given using the property *has cloud cover*; the altitude of the cloud layer is given by the property *has cloud altitude*. Sub-concepts are *Clear sky*, *Partly cloudy*, *Mostly cloudy*, *Overcast*, and *Unknown cloud cover*. 69–71, 103, 104, 134, *see* Weather phenomenon, Has cloud cover, Has cloud altitude, Okta, Clear sky, Partly cloudy, Mostly cloudy, Overcast *and* Unknown cloud cover

Cloudy weather
Sub-concept of *Weather state* representing a *Weather state* that is linked to an instance of *Mostly cloudy* or *Overcast* via the property *has weather phenomenon*. 69, *see* Weather state, Has weather phenomenon, Mostly cloudy *and* Overcast

Cold
Sub-concept of *Temperature* representing a temperature of more than or equal to 0 and less than 10 °C. 68, 69, 73, 106, *see* Temperature

Cold weather
Sub-concept of *Weather state* representing a *Weather state* that is linked to an instance of *Cold* or *Frost* via the property *has weather phenomenon*. 69, *see* Weather state, Has weather phenomenon, Cold *and* Frost

Current weather report
Sub-concept of *Weather report* describing the current weather. 68, 69, 81, 91, 106, 107, 134, *see* Weather report

Current weather report from sensor
Sub-concept of *Current weather report*; compiled from data from a *sensor source*. 68, 69, 107, *see* Weather report, Current weather report *and* Sensor source

Current weather report from service
Sub-concept of *Current weather report*; compiled from data from a *service source*. 68, 69, 107, *see* Weather report, Current weather report *and* Service source

D

Day
Sub-concept of *sun position* that represents the sun being exactly at or above the horizon. 69, 73, 105, *see* Sun position

Dew point
Sub-concept of *Weather phenomenon* that describes the dew point. The value is given using the property *has dew point value*. 68, 69, 75, 101, 103, *see* Weather phenomenon *and* Has dew point value

Directional wind
Sub-concept of *Wind* including a direction. 69, 73, 106, *see* Wind, North wind, East wind, South wind *and* Wind wind

Dry

Sub-concept of *Humidity* describing relative humidity of at least 30 and less than 40 percent. 69, 72, 104, *see* Humidity

Dry weather

Sub-concept of *Weather state* representing a *Weather state* that is linked to an instance of *Dry* or *Very dry* via the property *has weather phenomenon*. 69, *see* Weather state, Has weather phenomenon, Dry *and* Very dry

E

East wind

Sub-concept of *Wind* representing wind approximately coming from the east, e.g. originating from a direction of at least 45 and less than 135 degrees. 69, 75, 106, *see* Wind *and* Directional wind

End time

The time a *Weather report* is valid until, given by an instance of the concept *Interval* from *OWL-Time* [95] that specifies the interval between the report's *observation time* and its *end time*. 80, 81, 134, *see* Weather report, Has end time, Start time, Observation time *and* Interval

Extremely heavy rain

Sub-concept of *Precipitation* representing a precipitation probability greater than 0 and an intensity of more than 50 and at most 100 mm/h. 69, 72, 105, *see* Precipitation

F

Fair weather

Sub-concept of *Weather state* representing a *Weather state* that is an instance of *Calm weather*, *Clear weather*, and *No rain weather* at the same time.. 69, *see* Weather state, Calm weather, Clear weather *and* No rain weather

Forecast weather report

Sub-concept of *Weather report* for some time after its *observation time*. 68, 69, 106, *see* Weather report, Observation time, Short range weather report, Medium range weather report *and* Long range weather report

Frost

Sub-concept of *Temperature* representing a temperature below 0 °C. 68, 69, 73, 106, *see* Temperature

H

Has cloud altitude
Property that specifies the cloud altitude of a cloud layer represented by an instance of *Cloud cover*. This property makes use of concepts and properties defined by the *Measurement Units Ontology* [97] to provide both a value and its unit. 103, *see* Cloud cover

Has cloud cover
Property that specifies the cloud coverage of a cloud layer represented by an instance of *Cloud cover*. This property makes use of concepts and properties defined by the *Measurement Units Ontology* [97] to provide both a value and its unit. 103, 104, *see* Cloud cover

Has condition
Object property that links instances of *Weather state* and *Weather condition*; does not have an inverse property. 69, 100, 102, *see* Weather state *and* Weather condition

Has dew point value
Property that specifies the dew point value of an instance of *Dew point*. This property makes use of concepts and properties defined by the *Measurement Units Ontology* [97] to provide both a value and its unit. 101, 103, *see* Dew point

Has end time
An object property of *weather report* that specifies the report's *end time*. 70, 80, 101, 102, *see* Weather report, End time, Has start time *and* Observation time

Has humidity value
Property that specifies the humidity value of an instance of *Humidity*. This property makes use of concepts and properties defined by the *Measurement Units Ontology* [97] to provide both a value and its unit. 101, 103, 104, *see* Humidity

Has next weather state
Property that links an instance of *Weather state* to the immediately succeeding instance concerning their *start times*, if such an instance exists. Both this property and its reverse property *has previous weather state* are functional properties. 70, 102, 108, *see* Weather state *and* Has previous weather state

Has observation time
An object property of *Weather report* that specifies the report's *observation time*. 70, 80, 101, 102, *see* Weather report *and* Observation time

Has precipitation intensity
Property that specifies the precipitation intensity of an instance of *Precipitation*. This property makes use of concepts and properties defined by the *Measurement Units Ontology* [97] to provide both a value and its unit. 101, 103–105, *see* Precipitation

Has precipitation probability
Property that specifies the precipitation probability of an instance of *Precipitation*. This

property makes use of concepts and properties defined by the *Measurement Units Ontology* [97] to provide both a value and its unit. 101, 103–105, *see* Precipitation

Has pressure value
Property that specifies the pressure value of an instance of *Atmospheric pressure*. This property makes use of concepts and properties defined by the *Measurement Units Ontology* [97] to provide both a value and its unit. 78, 101, 103, 104, *see* Atmospheric pressure

Has previous weather state
Property that links an instance of *Weather state* to the immediately preceding instance concerning their *start times*, if such an instance exists. Both this property and its reverse property *has next weather state* are functional properties. 70, 102, *see* Weather state *and* Has next weather state

Has priority
A data property of *weather report* that specifies the report's *priority*. 70, 101, *see* Weather report *and* Priority

Has solar radiation value
Property that specifies the solar radiation value of an instance of *Solar radiation*. This property makes use of concepts and properties defined by the *Measurement Units Ontology* [97] to provide both a value and its unit. 101, 103, 105, *see* Solar radiation

Has source
Object property that links instances of *Weather report* and *Weather source*; inverse property of *is source of*. 69, 75, 100–102, 107, *see* Weather report, Weather source *and* Is source of

Has start time
An object property of *weather report* that specifies the report's *start time*. 70, 75, 80, 101, 102, 106, 107, *see* Weather report, Start time, Has end time *and* Observation time

Has sun direction
Property that specifies the sun's direction of an instance of *Sun position*. This property makes use of concepts and properties defined by the *Measurement Units Ontology* [97] to provide both a value and its unit. 101, 103, 105, *see* Sun position

Has sun elevation angle
Property that specifies the sun's elevation angle (the sun's height above horizon) of an instance of *Sun position*. This property makes use of concepts and properties defined by the *Measurement Units Ontology* [97] to provide both a value and its unit. 101, 103, 105, *see* Sun position

Has temperature value
Property that specifies the temperature value of an instance of *Temperature*. This property makes use of concepts and properties defined by the *Measurement Units Ontology* [97] to provide both a value and its unit. 81, 82, 93, 101, 103, 106, 134, *see* Temperature

Has weather phenomenon

Object property that links instances of *Weather state* and *Weather phenomenon*; inverse property of *belongs to state*. 69, 83, 100–102, *see* Weather state, Weather phenomenon *and* Belongs to state

Has weather state

Object property that links instances of *Weather report* and *Weather state*; inverse property of *belongs to weather report*. 69, 83, 100–102, *see* Weather report, Weather state *and* Belongs to weather report

Has wind direction

Property that specifies the wind direction of an instance of *Wind*. This property makes use of concepts and properties defined by the *Measurement Units Ontology* [97] to provide both a value and its unit. 73, 101, 103, 106, *see* Wind

Has wind speed

Property that specifies the wind speed of an instance of *Wind*. This property makes use of concepts and properties defined by the *Measurement Units Ontology* [97] to provide both a value and its unit. 101, 103, 106, *see* Wind

Heat

Sub-concept of *Temperature* representing a temperature above 30 °C. 68, 69, 73, 106, *see* Temperature

Heavy rain

Sub-concept of *Precipitation* representing a precipitation probability greater than 0 and an intensity of more than 20 and at most 50 mm/h. 69, 72, 104, *see* Precipitation

High pressure

Sub-concept of *Atmospheric pressure* representing a pressure of at least 1018 and less than 1028 hPa. 68, 70, 78, 104, *see* Atmospheric pressure

High radiation

Sub-concept of *Solar radiation* describing solar radiation of at least 500 W/m^2 and less than 750 W/m^2. 69, 73, 105, *see* Solar radiation

Hot weather

Sub-concept of *Weather state* representing a *Weather state* that is linked to an instance of *Heat* via the property *has weather phenomenon*. 69, *see* Weather state, Has weather phenomenon *and* Heat

Humidity

Sub-concept of *Weather phenomenon* representing the relative humidity of the air; the humidity value is specified using the property *has humidity value*. Sub-concepts are *Very dry*, *Dry*, *Normal humidity*, *Moist*, and *Very moist*. 68, 69, 72, 101, 103, 104, 134, *see* Weather phenomenon, Has humidity value, Very dry, Dry, Normal humidity, Moist *and* Very moist

Hurricane
Sub-concept of *Wind* representing wind with a speed of at least 32 m/s. 69, 75, 106, *see* Wind

I

Instant
Sub-concept of *Temporal entity* defined by *OWL-Time* [95] that specifies an instant. 20, 80, 86, 88, 102, *see* Temporal entity *and* Interval

Interval
Sub-concept of *Temporal entity* defined by *OWL-Time* [95] that specifies a period of time by its length (not start and end time). 20, 80, 88, 102, *see* Temporal entity *and* Instant

Is source of
Object property that links instances of *Weather report* and *Weather source*; inverse property of *has source*. 69, 100–102, *see* Weather report, Weather source *and* Has source

L

Lat
Property defined by the *Basic Geo (WGS84 lat/long) Vocabulary* [91] specifying the latitude of the *point* a *weather report* is valid for, given as a decimal number. Positive values refer to positions north of the equator, negative values refer to positions south of the equator. 19, 103, *see* Weather report *and* Point

Light rain
Sub-concept of *Precipitation* representing a precipitation probability greater than 0 and an intensity of more than 0 and at most 5 mm/h. 69, 72, 104, *see* Precipitation

Light wind
Sub-concept of *Wind* representing wind with a speed of at least 1 and less than 10 m/s. 69, 75, 106, *see* Wind

Location
An object property of *weather report* the geographical position the *weather report* is valid for, given by *altitude*, *longitude* and *latitude*, using the *WGS84* reference model [92]; links to an instance of the concept *point* of the *Basic Geo (WGS84 lat/long) Vocabulary* [91]. 70, 80, 82, 101–103, 134, *see* Weather report, Lat, Long *and* Alt

Long
Property defined by the *Basic Geo (WGS84 lat/long) Vocabulary* [91] specifying the longitude of the *point* a *weather report* is valid for, given as a decimal number. Positive values refer to positions east of the prime meridian at Greenwich, negative values refer to positions west of the prime meridian. 19, 103, *see* Weather report *and* Point

Long range weather report

Sub-concept of *Weather report* describing a forecast for a point of time at least 12 hours in the future relative to the *observation time*. Its sub-concepts are *Forecast 15 hours weather report*, *Forecast 18 hours weather report*, *Forecast 21 hours weather report*, and *Forecast 24 hours weather report*. 68, 69, 75, 76, 106, *see* Weather report, Forecast weather report *and* Observation time

Low pressure

Sub-concept of *Atmospheric pressure* representing a pressure of at least 998 and less than 1008 hPa. 68, 70, 78, 104, *see* Atmospheric pressure

Low radiation

Sub-concept of *Solar radiation* describing solar radiation of more than 0 W/m^2 and less than 250 W/m^2. 69, 72, 105, *see* Solar radiation

M

Medium radiation

Sub-concept of *Solar radiation* describing solar radiation of at least 250 W/m^2 and less than 500 W/m^2. 69, 72, 105, *see* Solar radiation

Medium rain

Sub-concept of *Precipitation* representing a precipitation probability greater than 0 and an intensity of more than 5 and at most 20 mm/h. 69, 72, 104, *see* Precipitation

Medium range weather report

Sub-concept of *Weather report* describing a forecast for a point of time more than 3 hours and less than 12 hours in the future relative to the *observation time*. Its sub-concepts are *Forecast 6 hours weather report*, *Forecast 9 hours weather report*, and *Forecast 12 hours weather report*. 68, 69, 75, 106, *see* Weather report, Forecast weather report *and* Observation time

Moist

Sub-concept of *Humidity* describing relative humidity of more than 70 and at most 80 percent. 69, 72, 104, *see* Humidity

Moist weather

Sub-concept of *Weather state* representing a *Weather state* that is linked to an instance of *Moist* or *Very moist* via the property *has weather phenomenon*. 69, *see* Weather state, Has weather phenomenon, Moist *and* Very moist

Mostly cloudy

Sub-concept of *Cloud cover* representing cloud coverage that is reported to be 5, 6, or 7 *okta*. 69, 70, 104, *see* Okta *and* Cloud cover

N

Nautical twilight
Sub-concept of *sun position* that represents the sun being more than 6 degrees and no more than 12 degrees below horizon. 69, 73, 105, *see* Sun position *and* Twilight

Night
Sub-concept of *sun position* that represents the sun being more than 18 degrees below horizon. 69, 73, 105, *see* Sun position

No radiation
Sub-concept of *Solar radiation* describing the absence of any solar radiation ($0\,W/m^2$). 69, 72, 105, *see* Solar radiation

No rain
Sub-concept of *Precipitation* representing absence precipitation (because either the intensity of precipitation is 0). 69, 72, 104, *see* Precipitation

No rain weather
Sub-concept of *Weather state* representing a *Weather state* that is linked to an instance of *No rain* via the property *has weather phenomenon*. 69

Normal humidity
Sub-concept of *Humidity* describing relative humidity of at least 40 and at most 70 percent. 69, 72, 104, *see* Humidity

North wind
Sub-concept of *Wind* representing wind approximately coming from the north, e.g. originating from a direction of at least 315 or less than 45 degrees. 69, 73, 106, *see* Wind *and* Directional wind

O

Observation time
The time when the weather data was collected from the weather sensor or the Internet weather service, given by an instance of the concept *Instant* from *OWL-Time* [95]. 80, 81, 134, *see* Weather report, Has observation time *and* Instant

Okta
Unit of measurement that specifies the amount of *cloud cover* on a range from 0 (*Clear sky*) to 8 (*Overcast*); a value of 9 represents an *Unknown cloud cover*. 103, *see* Cloud cover, Clear sky, Partly cloudy, Mostly cloudy, Overcast *and* Unknown cloud cover

Overcast
Sub-concept of *Cloud cover* representing cloud coverage that is reported to be 8 *okta*. 69, 70, 104, *see* Okta *and* Cloud cover

P

Partly cloudy
Sub-concept of *Cloud cover* representing cloud coverage that is reported to be 1, 2, 3, or 4 *okta*. 69, 70, 104, *see* Okta *and* Cloud cover

Pleasant temperature weather
Sub-concept of *Weather state* representing a *Weather state* that is linked to an instance of *Below room temperature*, *Room temperature*, or *Above room temperature* via the property *has weather phenomenon*. 69, *see* Weather state, Has weather phenomenon, Below room temperature, Room temperature *and* Above room temperature

Point
A concept defined by the *Basic Geo (WGS84 lat/long) Vocabulary* [91] representing a location which is given by *altitude*, *longitude* and *latitude*. It is connected from an entity using the property *location*. 88, 102, *see* Lat, Long, Alt *and* Location

Precipitation
Sub-concept of *Weather phenomenon* that describes precipitation (intensity and probability). Intensity is specified by the property *has precipitation intensity*, probability by *has precipitation probability*. Sub-concepts are *No rain*, *Light rain*, *Medium rain*, *Heavy rain*, *Extremely heavy rain*, and *Tropical storm rain*. 69, 72, 101, 103–105, 134, *see* Weather phenomenon, Has precipitation intensity, Has precipitation probability, No rain, Light rain, Medium rain, Heavy rain, Extremely heavy rain *and* Tropical storm rain

Priority
An integer value indicating which *weather report* for a certain period of time is to be preferred over another *weather report* for the same period of time. *see* Weather report *and* Has priority

R

Rainy weather
Sub-concept of *Weather state* representing a *Weather state* that is linked to an instance of *Light rain*, *Medium rain*, *Heavy rain*, *Extremely heavy rain*, or *Tropical storm rain* via the property *has weather phenomenon*. 69, *see* Weather state, Has weather phenomenon, Light rain, Medium rain, Heavy rain, Extremely heavy rain *and* Tropical storm rain

Room temperature
Sub-concept of *Temperature* representing a temperature of least 20 and at most 25 °C. 69, 73, 106, *see* Temperature

S

Sensor source
A sub-concept of *Weather Source* representing a sensor or a set of sensors offering some kind of weather data. 68, 69, 76, 88, 100, 107, *see* Weather source

Service source
A sub-concept of *Weather Source* representing an Internet weather service offering some kind of weather data. 68, 69, 76, 88, 100, 107, *see* Weather source

Severe weather
Sub-concept of *Weather state* representing a *Weather state* that is an instance of *Stormy weather* and *Very rainy weather* at the same time. 69, *see* Weather state, Stormy weather *and* Very rainy weather

Short range weather report
Sub-concept of *Weather report* describing a forecast for a point of time at most 3 hours in the future relative to the *observation time*. Its sub-concepts are *Forecast 1 hour weather report*, *Forecast 2 hours weather report*, and *Forecast 3 hours weather report*. 68, 69, 75, 106, *see* Weather report, Forecast weather report *and* Observation time

Solar radiation
Sub-concept of *Weather phenomenon* representing solar radiation; the value is specified using the property *has solar radiation value*. Sub-concepts are *No radiation*, *Low radiation*, *Mediumr radiation*, *High radiation*, and *Very high radiation*. 69, 72, 73, 101, 103, 105, 134, *see* Weather phenomenon, Has solar radiation value, No radiation, Low radiation, Medium radiation, High radiation *and* Very high radiation

Solar twilight
Sub-concept of *sun position* that represents the sun being above horizon, but less than 6 degrees above. 69, 73, 105, *see* Sun position *and* Day

South wind
Sub-concept of *Wind* representing wind approximately coming from the south, e.g. originating from a direction of at least 135 and less than 225 degrees. 69, 75, 106, *see* Wind *and* Directional wind

Start time
The time a *Weather report* is valid from, given by an instance of the concept *Interval* from *OWL-Time* [95] that specifies the interval between the report's *observation time* and its *start time*. 68, 80, 81, 134, *see* Weather report, Has start time, End time, Observation time *and* Interval

Storm
Sub-concept of *Wind* representing wind with a speed of at least 20 m/s. 69, 75, 106, *see* Wind

Stormy weather
Sub-concept of *Weather state* representing a *Weather state* that is linked to an instance of *Storm* or *Hurricane* via the property *has weather phenomenon*. *see* Weather state, Has weather phenomenon, Storm *and* Hurricane

Strong wind
Sub-concept of *Wind* representing wind with a speed of at least 10 and less than 20 m/s. 69, 75, 106, *see* Wind

Sun below horizon
Sub-concept of *sun position* that represents the sun being below the horizon. 69, 73, 105, *see* Sun position, Night *and* Twilight

Sun from east
Sub-concept of *sun position* that represents a sun direction of more than 45 and at most 135 degrees. 69, 73, 105, *see* Sun position

Sun from north
Sub-concept of *sun position* that represents a sun direction of more than 315 or at most 45 degrees. 69, 73, 105, *see* Sun position

Sun from south
Sub-concept of *sun position* that represents a sun direction of more than 135 and at most 225 degrees. 69, 73, 105, *see* Sun position

Sun from west
Sub-concept of *sun position* that represents a sun direction of more than 225 and at most 315 degrees. 69, 105, *see* Sun position

Sun position
Sub-concept of *Weather phenomenon* describing the position of the sun, given by its elevation above horizon (specified by the property *has sun elevation angle* and its direction *has sun direction*. Sub-concepts are *Day*, *Solar twilight*, *Twilight*, *Civil twilight*, *Nautical twilight*, *Astronomical twilight*, and *Night* for the elevation above horizon, and *Sun from north*, *Sun from east*, *Sun from South*, and *Sun from West* for the direction.. 69, 73, 74, 101, 103, 105, 134, *see* Weather phenomenon, Has sun elevation angle, Has sun direction, Day, Solar twilight, Twilight, Civil twilight, Nautical twilight, Astronomical twilight, Night, Sun from north, Sun from east, Sun from south *and* Sun from west

Sun protection weather
Sub-concept of *Weather state* representing a *Weather state* that is an instance of *Clear weather* and linked to an instance of *Day* via the property *has weather phenomenon*. 69

T

Temperature
Sub-concept of *Weather phenomenon* representing temperature. The temperature value is specified using the property *has temperature value*. Sub-concepts are *Frost*, *Cold*, *Below room temperature*, *Room temperature*, *Above room temperature*, and *Heat*. 68, 69, 73, 74, 80–82, 93, 101, 103, 106, 134, *see* Weather phenomenon, Has temperature value, Frost, Cold, Below room temperature, Room temperature, Above room temperature *and* Heat

Temporal entity
Concept defined by *OWL-Time* [95]; either an *Instant* or an *Interval*. 20, 80, 88, *see* Instant *and* Interval

Thunderstorm
Sub-concept of *Weather state* representing a *Weather state* that is an instance of *Severe weather* and is linked to the instance *Thunder* of the concept *Weather condition* via the property *has condition* at the same time. 69, *see* Weather state, Has condition, Weather condition *and* Severe weather

Tropical storm rain
Sub-concept of *Precipitation* representing a precipitation probability greater than 0 and an intensity of more than 100 mm/h. 69, 72, 105, *see* Precipitation

Twilight
Sub-concept of *sun position* that represents the sun being below horizon and at most 18 degrees below horizon. 69, 73, 105, *see* Sun position, Civil twilight, Nautical twilight *and* Astronomical twilight

U

Unknown cloud cover
Sub-concept of *Cloud cover* representing cloud coverage that is reported to be 9 *okta*. 69, 70, 104, *see* Okta *and* Cloud cover

V

Very dry
Sub-concept of *Humidity* describing relative humidity of less than 30 percent. 69, 72, 104, *see* Humidity

Very high pressure
Sub-concept of *Atmospheric pressure* representing a pressure of at least 1028 hPa. 68, 70, 78, 104, *see* Atmospheric pressure

Very high radiation
Sub-concept of *Solar radiation* describing solar radiation of at least 750 W/m^2. 69, 73, 105, *see* Solar radiation

Very low pressure
Sub-concept of *Atmospheric pressure* representing a pressure of less than 998 hPa. 68, 70, 78, 104, *see* Atmospheric pressure

Very moist
Sub-concept of *Humidity* describing relative humidity of more than 80 percent. 69, 72, 104, *see* Humidity

Very rainy weather
Sub-concept of *Weather state* representing a *Weather state* that is linked to an instance of *Heavy rain*, *Extremely heavy rain*, or *Tropical storm rain* via the property *has weather phenomenon*. By reasoning, it is a sub-concept of *Rainy weather* as well. 69, *see* Weather state, Rainy weather, Heavy rain, Extremely heavy rain *and* Tropical storm rain

W

Weather condition
A concept describing the overall state of the weather, represented by (a combination of) these individuals: *Sun*, *Light clouds*, *Partly cloudy*, *Cloudy*, *Fog*, *Rain*, *Snow*, *Sleet*, and *Thunder*. 67–70, 79, 88, 99, 100, 102, 107, 134, 135, *see* Weather state *and* Has condition

Weather phenomenon
A concept that represents a certain weather element. Relevant weather elements are *Atmospheric pressure*, *Cloud cover*, *Dew point*, *Humidity*, *Precipitation*, *Solar radiation*, *Sun position*, *Temperature*, and *Wind*. 67–71, 75, 76, 78, 79, 83, 88, 93, 96, 99, 101, 102, 134, 135, *see* Atmospheric pressure, Cloud cover, Dew point, Humidity, Precipitation, Solar radiation, Sun position, Temperature *and* Wind

Weather report
Concept that summarizes all data acquired at a certain *observation time* for a certain *location* from a *weather source*, about the current weather or the weather some time in the future. Exactly one *weather state* is linked to each *weather report*. 67–70, 75, 76, 78–83, 88, 90, 91, 96, 99–102, 106–108, 134, 135, *see* Weather state, Current weather report, Forecast weather report, Weather source, Observation time, Start time, End time *and* Location

Weather report from sensor
Sub-concept of *weather report*; contains data obtained from one or more *sensor sources*. 68, 69, 107, *see* Weather report *and* Sensor source

Weather report from service
Sub-concept of *weather report*; contains data obtained from an Internet *service source*. 68, 69, 107, *see* Weather report *and* Service source

Weather source
A concept representing a source for weather data, either a set of *weather sensors* or an Internet *weather service*. Sub-concepts are *Sensor source*, and *Service source*. 69, 76, 78, 88, 99, 100, 102, 134, 135, *see* Sensor source *and* Service source

Weather state
A concept that represents all data available about weather phenomena for a certain *Weather report*. A set of instances of the concept *Weather phenomenon* are linked to every instance of *Weather state*. 67–70, 76–80, 83, 88, 96, 99, 100, 102, 134, 135, *see* Weather report *and* Weather phenomenon

Wind

Sub-concept of *Weather phenomenon* representing wind (speed and direction, the latter is optional). Wind speed is given using the property *has wind speed*; wind direction is specified using *has wind direction*. Sub-concepts are *Directional wind*, *North wind*, *East wind*, *South wind*, and *West wind* for the wind direction, and *Calm*, *Light wind*, *Strong wind*, *Storm*, and *Hurricane* for the wind speed. 69, 73, 75, 101, 103, 106, 134, *see* Weather phenomenon, Has wind speed, Has wind direction, Directional wind, North wind, East wind, South wind, Wind wind, Calm, Light wind, Strong wind, Storm *and* Hurricane

Wind wind

Sub-concept of *Wind* representing wind approximately coming from the wind, e.g. originating from a direction of at least 225 and less than 315 degrees. 69, 106, *see* Wind *and* Directional wind

Windy weather

Sub-concept of *Weather state* representing a *Weather state* that is either an instance of *Stormy weather* or linked to an instance of *Strong wind* via the property *has weather phenomenon*. 69, *see* Weather state, Has weather phenomenon, Stormy weather *and* Strong wind

APPENDIX C

Acronyms

ADDS Aviation Digital Data Service

AMS American Meteorological Society

API Application Programming Interface

ASCII American Standard Code for Information Interchange

BACnet Building automation and control networks

BAS Building Automation Systems

CSV Comma-separated values

DAML *DARPA* Agent Markup Language

DARPA Defense Advanced Research Projects Agency

DOAP Description of a Project

DOM Document Object Model

DWD Deutscher Wetterdienst (“German weather service”)

EHS European Home Systems Protocol

EIB European Installation Bus

FAA Federal Aviation Administration

FOAF Friend of a Friend

FTP File Transfer Protocol

HTTP Hypertext Transfer Protocol

HVAC Heating, ventilation and air conditioning

ICAO International Civil Aviation Organization

JSON JavaScript Object Notation

JSONP *JSON* with padding

KIF Knowledge Interchange Format

LCN Local Control Network

LKIF Legal Knowledge Interchange Format

METAR Meteorological Aerodrome Report

MIT Massachussets Institute of Technology

MSL Mean sea level

MUO Measurements Units Ontology

N3 Notation3

NASA National Aeronautics and Space Administration

NextGen Next Generation Air Transportation System

NNEW Next Generation Network Enabled Weather

NOAA National Oceanic and Atmospheric Administration

NWS National Weather Service

OASIS Organization for the Advancement of Structured Information Standards

OBO The Open Biological and Biomedical Ontologies

OGC Open Geospatial Consortium

OIL Ontology Interchange Language

OM Ontology of Units of Measure and Related Concepts

OWL Web Ontology Language

PATO Phenotypic Quality Ontology

PLC Programmable Logic Controller

QUDT Quantities, Units, Dimensions and Data Types in OWL and XML

QUOMOS *OASIS* Quantities and Units of Measure Ontology Standard

RDF Resource Description Framework

RDFS *RDF* schema

REST Representational State Transfer

RFC Request for Comments

RSS Rich Site Summary

SAX Simple *API* for XML

SIOC Semantically-Interlinked Online Communities

SKOS Simple Knowledge Organization System

SOAP Simple Object Access Protocol

SPARQL *SPARQL* Protocol and *RDF* Query Language

SSN Semantic Sensor Network

SSN-XG *W3C* Semantic Sensor Network Incubator group

SSW Semantic Sensor Web

SWE Sensor Web Enablement

SWEET Semantic Web for Earth and Environmental Terminology

SWIG *W3C* Semantic Web Interest Group

SYNOP Surface Synoptic Observations

TC Technical Committee

TOVE TOronto Visual Enterprise

Turtle Terse *RDF* Triple Language

UMBEL Upper Mapping and Binding Exchange Layer

UML Unified Modeling Language

UPON Unified Process for ONtology building

URI Uniform Resource Identifier

URL Uniform Resource Locator

W3C World Wide Web Consortium

WGS84 World Geodetic System 1984

XML Extensible Markup Language

List of Figures

List of Tables

Bibliography

[1] Todo insert title (web page currently unavailable). http://www.its-rtc.com/.

[2] Nicole King. Smart home – a definition. 2003.

[3] *Smart home research*, volume 2, 2004.

[4] M. C. Mozer. Lessons from an adaptive house. In M. Cook and R. Das, editors, *Smart environments: Technologies, protocols, and applications*, pages 273–294. J. Wiley & Sons.

[5] Julie A. Kientz, Shwetak N. Patel, Brian Jones, Ed Price, Elizabeth D. Mynatt, and Gregory D. Abowd. The georgia tech aware home. In *CHI '08 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '08, pages 3675–3680, New York, NY, USA, 2008. ACM.

[6] S. Helal, W. Mann, H. El-Zabadani, J. King, Y. Kaddoura, and E. Jansen. The gator tech smart house: a programmable pervasive space. *Computer*, 38(3):50–60, 2005.

[7] Tiiu Koskela and Kaisa Väänänen-Vainio-Mattila. Evolution towards smart home environments: empirical evaluation of three user interfaces. *Personal Ubiquitous Comput.*, 8(3-4):234–240, July 2004.

[8] S.S. Intille. Designing a home of the future. *Pervasive Computing, IEEE*, 1(2):76–82, 2002.

[9] A.J. Bernheim Brush, Bongshin Lee, Ratul Mahajan, Sharad Agarwal, Stefan Saroiu, and Colin Dixon. Home automation in the wild: challenges and opportunities. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 2115–2124, New York, NY, USA, 2011. ACM.

[10] Shaun Salzberg. home maestro.

[11] W. Keith Edwards, Rebecca E. Grinter, Ratul Mahajan, and David Wetherall. Advancing the state of home networking. *Commun. ACM*, 54(6):62–71, June 2011.

[12] Institute of computer aided automation, automation systems group, thinkhome – overview. https://www.auto.tuwien.ac.at/projectsites/thinkhome/overview.html.

[13] Christian Reinisch, Mario J. Kofler, Felix Iglesias, and Wolfgang Kastner. ThinkHome: Energy Efficiency in Future Smart Homes. *EURASIP Journal on Embedded Systems*, 2011, 2011.

[14] Christian Reinisch, Mario J. Kofler, and Wolfgang Kastner. ThinkHome: A Smart Home as Digital Ecosystem. In *Proceedings of 4th IEEE International Conference on Digital Ecosystems and Technologies (DEST '10)*, pages 256–261, April 2010.

[15] Samuel Prívara, Jan Široký, Lukáš Ferkl, and Jiří Cigler. Model predictive control of a building heating system: The first experience. *Energy and Buildings*, 43(2–3):564 – 572, 2011.

[16] Petru-Daniel Moroşan, Romain Bourdais, Didier Dumur, and Jean Buisson. Building temperature regulation using a distributed model predictive control. *Energy and Buildings*, 42(9):1445 – 1452, 2010.

[17] Frauke Oldewurtel, Alessandra Parisio, Colin N. Jones, Dimitrios Gyalistras, Markus Gwerder, Vanessa Stauch, Beat Lehmann, and Manfred Morari. Use of model predictive control and weather forecasts for energy efficient building climate control. *Energy and Buildings*, 45(0):15 – 27, 2012.

[18] Dieter Fensel. *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2 edition, 2003.

[19] W3C OWL Working Group. OWL 2 Web Ontology Language Document Overview (Second Edition). W3C Recommendation, 11 December 2012. http://www.w3.org/TR/2012/REC-owl2-overview-20121211/. Latest version available at http://www.w3.org/TR/owl2-overview/.

[20] I. Horrocks, O. Kutz, and U. Sattler. The even more irresistible sroiq. In *Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 57–67.

[21] Jon Barwise. An introduction to first-order logic. In Jon Barwise, editor, *HANDBOOK OF MATHEMATICAL LOGIC*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, pages 5 – 46. Elsevier, 1977.

[22] V. Richard Benjamins, Dieter Fensel, Stefan Decker, and ASUNCIÓN GÓMEZ PÉREZ. (ka)□: Building ontologies for the internet: a mid term report. *INTERNATIONAL JOURNAL OF HUMAN COMPUTER STUDIES*, 51:687–712, 1999.

[23] Elena Simperl. Reusing ontologies on the semantic web: A feasibility study. *Data Knowl. Eng.*, 68(10):905–925, October 2009.

[24] Michael Genesereth, Richard E. Fikes, Ronald Brachman, Thomas Gruber, Patrick Hayes, Reed Letsinger, Vladimir Lifschitz, Robert Macgregor, John Mccarthy, Peter Norvig, and Ramesh Patil. Knowledge interchange format version 3.0 reference manual, 1992.

[25] Dan Connolly, Frank van Harmelen, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. DAML+OIL (March 2001) Reference Description. W3C Note, 18 December 2001. http://www.w3.org/TR/2001/NOTE-daml+oil-reference-20011218. Latest version available at http://www.w3.org/TR/daml+oil-reference.

[26] The darpa agent markup language homepage. http://www.daml.org/.

[27] Dieter Fensel, Frank van Harmelen, Ian Horrocks, Deborah L. McGuinness, and Peter F. Patel-Schneider. Oil: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16(2):38–45, March 2001.

[28] Dan Bickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, 10 February 2004. http://www.w3.org/TR/2004/REC-rdf-schema-20040210/. Latest version available at http://www.w3.org/TR/rdf-schema/.

[29] edumbill / doap – rdf schema for describing software projects. https://github.com/edumbill/doap/.

[30] Dublin core®metadata initiative. http://dublincore.org/.

[31] J. Kunze and T. Baker. The Dublin Core Metadata Element Set. RFC 5013 (Informational), August 2007.

[32] The Friend of a Friend (FOAF) project. http://www.foaf-project.org/.

[33] Dan Brickley and Libby Miller. FOAF Vocabulary Specification 0.98. Namespace Document 9 August 2010 – Marco Polo Edition. http://xmlns.com/foaf/spec/20100809.html. Latest version available at http://xmlns.com/foaf/spec/.

[34] Diego Berrueta, Dan Brickley, Stefan Decker, Sergio Fernández, Christoph Görn, Andreas Harth, Tom Heath, Kingsley Idehen, Kjetil Kjernsmo, Alistair Miles, Alexandre Passant, Axel Polleres, Luis Polo, and Michael Sintek. SIOC Core Ontology Specification. W3C Member Submission 12 June 2007. http://www.w3.org/Submission/2007/SUBM-sioc-spec-20070612/. Latest version available at http://www.w3.org/Submission/sioc-spec/.

[35] SKOS Simple Knowledge Organization System - Home Page. http://www.w3.org/2004/02/skos/.

[36] Alistair Miles and Sean Bechhofer. SKOS Simple Knowledge Organization System Reference. W3C Recommendation 18 August 2009. http://www.w3.org/TR/2009/REC-skos-reference-20090818/. Latest version available at http://www.w3.org/TR/skos-reference.

[37] UMBEL Web Site. http://umbel.org/.

[38] Adam Pease, Ian Niles, and John Li. The suggested upper merged ontology: A large ontology for the semantic web and its applications. In *In Working Notes of the AAAI-2002 Workshop on Ontologies and the Semantic Web*, page 2002, 2002.

[39] *W3C* Semantic Web Activity Homepage. http://www.w3.org/2001/sw/.

[40] World Wide Web Consortium (*W3C*). http://www.w3.org/.

[41] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, May 2001.

[42] Graham Klyne and Jeremy J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation, 10 February 2004. `http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/`. Latest version available at `http://www.w3.org/TR/rdf-concepts/`.

[43] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (INTERNET STANDARD), January 2005.

[44] Paul V. Biron and Ashok Malhotra. W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures. W3C Recommendation 5 April 2012. `http://www.w3.org/TR/2012/REC-xmlschema11-1-20120405/`. Latest version available at `http://www.w3.org/TR/xmlschema11-1/`.

[45] Paul V. Biron and Ashok Malhotra. XML Schema Part 2: Datatypes Second Edition. W3C Recommendation, 28 October 2004. `http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/`. Latest version available at `http://www.w3.org/TR/xmlschema-2/`.

[46] Dave Beckett. RDF/XML Syntax Specification (Revised). W3C Recommendation, 10 February 2004. `http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/`. Latest version available at `http://www.w3.org/TR/rdf-syntax-grammar/`.

[47] Tim Berners-Lee and Dan Connolly. Notation3 (N3): A readable RDF syntax. W3C Team Submission 28 March 2011. `http://www.w3.org/TeamSubmission/2011/SUBM-n3-20110328/`. Latest version available at `http://www.w3.org/TeamSubmission/n3/`.

[48] Beckett, Dave and Berners-Lee, Tim and Prud'hommeaux, Eric and Carothers, Gavin. Turtle. Terse RDF Triple Language. W3C Candidate Recommendation (work in progress), 19 February 2013. `http://www.w3.org/TR/2013/CR-turtle-20130219/`. Latest version available at `http://www.w3.org/TR/rdf-syntax-grammar/`.

[49] Patrick Hayes. RDF Semantics. W3C Recommendation, 10 February 2004. `http://www.w3.org/TR/2004/REC-rdf-mt-20040210/`. Latest version available at `http://www.w3.org/TR/rdf-mt/`.

[50] OWL Working Group. http://www.w3.org/2007/OWL/wiki/OWL_Working_Group.

[51] Jacobs, Ian. World Wide Web Consortium Process Document, 14 October 2005. http://www.w3.org/2005/10/Process-20051014/. Latest version available at http://www.w3.org/Consortium/Process/.

[52] W3C OWL Working Group. OWL Web Ontology Language Overview. W3C Recommendation 10 February 2004. http://www.w3.org/TR/2004/REC-owl-features-20040210/. Latest version available at http://www.w3.org/TR/owl-features/.

[53] Stefano Mazzocchi. Closed World vs. Open World: the First Semantic Web Battle. http://www.betaversion.org/ stefano/linotype/news/91/, jun 2005.

[54] A. Elci, B. Rahnama, and S. Kamran. Defining a strategy to select either of closed/open world assumptions on semantic robots. In *Computer Software and Applications, 2008. COMPSAC '08. 32nd Annual IEEE International*, pages 417–423, 2008.

[55] Raffaella Bernardi. Monotonic reasoning from a proof-theoretic perspective, 1999.

[56] The W3C SPARQL Working Group. SPARQL 1.1 Overview. W3C Recommendation, 21 March 2013. http://www.w3.org/TR/2013/REC-sparql11-overview-20130321/. Latest version available at http://www.w3.org/TR/sparql11-overview/.

[57] The Protégé Ontology Editor and Knowledge Acquisition System. http://protege.stanford.edu/.

[58] Pellet: Owl 2 reasoner for java. http://clarkparsia.com/pellet/.

[59] RacerPro. http://semanticweb.org/wiki/RacerPro.

[60] OWL : FaCT++. http://owl.man.ac.uk/factplusplus/.

[61] HermiT OWL Reasoner. http://www.hermit-reasoner.com/.

[62] Apache jena. http://jena.apache.org/.

[63] OpenLink Virtuoso Universal Server. http://virtuoso.openlinksw.com/.

[64] Michael Wooldridge. *An Introduction to MultiAgent Systems*. Wiley Publishing, 2nd edition, 2009.

[65] Jade – Java Agent DEvelopment Framework. http://jade.tilab.com/.

[66] JACK®. http://aosgrp.com/products/jack/.

[67] M. Presser, P.M. Barnaghi, M. Eurich, and C. Villalonga. The sensei project: integrating the physical world with the digital world of the network of the future. *Communications Magazine, IEEE*, 47(4):1–4, 2009.

[68] Anil Aswani, Neal Master, Jay Taneja, Andrew Krioukov, David E. Culler, and Claire Tomlin. Energy-efficient building hvac control using hybrid system lbmpc. *CoRR*, abs/1204.4717, 2012.

[69] F. Oldewurtel, A. Parisio, C.N. Jones, M. Morari, D. Gyalistras, M. Gwerder, V. Stauch, B. Lehmann, and K. Wirth. Energy efficient building climate control using stochastic model predictive control and weather predictions. In *American Control Conference (ACC), 2010*, pages 5100–5105, 2010.

[70] Mike Botts, George Percivall, Carl Reed, and John Davidson. Ogc® sensor web enablement: Overview and high level architecture. In Silvia Nittel, Alexandros Labrinidis, and Anthony Stefanidis, editors, *GeoSensor Networks*, volume 4540 of *Lecture Notes in Computer Science*, pages 175–190. Springer Berlin Heidelberg, 2008.

[71] Open Geospatial Consortium | *OGC*®. http://www.opengeospatial.org/.

[72] A. Sheth, C. Henson, and S.S. Sahoo. Semantic sensor web. *Internet Computing, IEEE*, 12(4):78 –83, July-Aug. 2008.

[73] K. Thirunarayan, C.A. Henson, and A.P. Sheth. Situation awareness via abductive reasoning from semantic sensor data: A preliminary report. In *Collaborative Technologies and Systems, 2009. CTS '09. International Symposium on*, pages 111–118, 2009.

[74] Hoan-Suk Choi and Woo-Seop Rhee. Distributed semantic sensor web architecture. In *TENCON 2012 - 2012 IEEE Region 10 Conference*, pages 1–6, 2012.

[75] *W3C* Semantic Sensor Network Incubator Group. http://www.w3.org/2005/Incubator/ssn/.

[76] Michael Compton, Payam Barnaghi, Luis Bermudez, Raul Garcıa-Castro, Oscar Corcho, Simon Cox, John Graybeal, Manfred Hauswirth, Cory Henson, Arthur Herzog, Vincent Huang, Krzysztof Janowicz, W. David Kelsey, Danh Le Phuoc, Laurent Lefort, Myriam Leggieri, Holger Neuhaus, Andriy Nikolov, Kevin Page, Alexandre Passant, Amit Sheth, and Kerry Taylor. The ssn ontology of the w3c semantic sensor network incubator group. *Web Semantics: Science, Services and Agents on the World Wide Web*, 17(0), 2012.

[77] Ontology:DOLCE+DnS Ultralite – odp. http://ontologydesignpatterns.org/wiki/Ontology:DOLCE+DnS_

[78] A. Sheth. Citizen sensing, social signals, and enriching human experience. *Internet Computing, IEEE*, 13(4):87–92, 2009.

[79] Ken Wenzel, Jörg Riegel, Andreas Schlegel, and Matthias Putz. Semantic web based dynamic energy analysis and forecasts in manufacturing engineering. In Jürgen Hesselbach and Christoph Herrmann, editors, *Glocalized Solutions for Sustainability in Manufacturing*, pages 507–512. Springer Berlin Heidelberg, 2011.

[80] Liang Yu and Yong Liu. Using the linked data approach in a heterogeneous sensor web: challenges, experiments and lessons learned.

[81] Robert G. Raskin and Michael J. Pan. Knowledge representation in the semantic web for Earth and environmental terminology (SWEET). *Computers & Geosciences*, 31(9):1119 – 1125, 2005. Application of XML in the Geosciences.

[82] SWEET Ontologies. http://sweet.jpl.nasa.gov/.

[83] Space, Stars, Mars, Earth, Planets and More – *NASA* Jet Propulsion Laboratory. http://www.jpl.nasa.gov/.

[84] Rob Raskin and Michael Pan. Semantic web for earth and environmental terminology (sweet. In *In Proc. of the Workshop on Semantic Web Technologies for Searching and Retrieving Scientific Data*, 2003.

[85] Peter Fox, Deborah McGuinness, Robert Raskin, and Krishna Sinha. A volcano erupts: semantically mediated integration of heterogeneous volcanic and atmospheric data. In *Proceedings of the ACM first workshop on CyberInfrastructure: information management in eScience*, CIMS '07, pages 1–6, New York, NY, USA, 2007. ACM.

[86] Jian Zhong, Atilla Aydina, and Deborah L. McGuinness. Ontology of fractures. *Journal of Structural Geology*, 31(3):251 – 259, 2009.

[87] R Ramachandran, S Graves, and R Raskin. Ontology re-engineering use case: Extending sweet to map climate and forecasting vocabulary terms. In *AGU Spring Meeting Abstracts*, volume 1, page 02, 2006.

[88] Fact Sheet - NextGen. Federal Aviation Authority, Feb. 2007.

[89] *FAA*: Home. http://www.faa.gov/.

[90] Aaron Braeckel. Nextgen network–enabled weather (nnew). *Briefing to NCAR and NOAA Staff*, 2009.

[91] Brickley, Dan. Basic Geo (WGS84 lat/long) Vocabulary, 06 February 2004. `http://www.w3.org/2003/01/geo/`.

[92] NIMA TR 8350.2. *Department of Defense World Geodetic System 1984: Its definition and relationships with local geodetic systems*. DMA technical report. National Imagery and Mapping Agency, third edition, 1997.

[93] Joshua Lieberman, Raj Singh, and Chris Goad. W3C Geospatial Vocabulary. W3C Incubator Group Report 23 October 2007. `http://www.w3.org/2005/Incubator/geo/XGR-geo-20071023/`. Latest version available at `http://www.w3.org/2005/Incubator/geo/XGR-geo/`.

[94] Joshua Lieberman, Raj Singh, and Chris Goad. W3C Geospatial Ontologies. W3C Incubator Group Report 23 October 2007. `http://www.w3.org/2005/Incubator/geo/XGR-geo-ont-20071023/`. Latest version available at `http://www.w3.org/2005/Incubator/geo/XGR-geo-ont/`.

[95] Hobbs, Jerry R. and Pan Feng. Time Ontology in OWL. W3C Working Draft, 27 September 2004. http://www.w3.org/TR/2006/WD-owl-time-20060927/. Latest version available at http://www.w3.org/TR/owl-time/.

[96] MUO – Measurement Units Ontology. Working Draft – DD April 2008. http://idi.fundacionctic.org/muo/muo-vocab.html.

[97] Polo, Luis and Berrueta, Diego. MUO – Measurement Units Ontology. Working Draft, April 2008. http://mymobileweb.morfeo-project.org/specs/name.

[98] Ontology of units of Measure (*OM*). http://www.wurvoc.org/vocabularies/om-1.6/.

[99] Hajo Rijgersberg, Mark van Assem, and Jan L. Top. Ontology of units of measure and related concepts. *Semantic Web*, 4(1):3–13, 2013.

[100] QUDT - Quantities, Units, Dimensions and Data Types in OWL and XML. September 11, 2011. http://www.qudt.org/.

[101] *OASIS* Quantities and Units of Measure Ontology Standard (*QUOMOS*) *TC*. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=quomos.

[102] The Open Biological and Biomedical Ontologies. http://obofoundry.org/.

[103] The Open Biological and Biomedical Ontologies – Metrical units for use in conjunction with *PATO*. http://obofoundry.org/cgi-bin/detail.cgi?id=unit.

[104] unit-ontology – Ontology of Units and Measurements. http://code.google.com/p/unit-ontology/.

[105] Natalya F. Noy and Deborah L. McGuinness. Ontology Development 101: A Guide to Creating Your First Ontology. Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880, March 2001.

[106] Mike Uschold and Martin King. Towards a methodology for building ontologies. In *In Workshop on Basic Ontological Issues in Knowledge Sharing, held in conjunction with IJCAI-95*, 1995.

[107] Michael Grüninger and Mark S. Fox. Methodology for the design and evaluation of ontologies. 1995.

[108] Mariano Fernandez-Lopez, Asuncion Gomez-Perez, and Natalia Juristo. Methontology: from ontological art towards ontological engineering. In *Proceedings of the AAAI97 Spring Symposium*, pages 33–40, Stanford, USA, March 1997.

[109] T. N. Palmer. Predicting uncertainty in forecasts of weather and climate. *Reports on Progress in Physics*, 63(2):71–116, 2000.

[110] Harvey Stern. The accuracy of weather forecasts for Melbourne, Australia. *Meteorological Applications*, 15:65–71, 2008.

[111] T.S. Glickman and American Meteorological Society. *Glossary of Meteorology*. University of Chicago Press, 2000.

[112] Andrew Chatha. Fieldbus: The foundation for field control systems. *Control Engineering*, 41(6):77–80, 1994.

[113] *IEC 61158 (1999-10): Digital data communication for measurement and control – Fieldbus for use in industrial control systems*. International Electrotechnical Commission, Geneve, Switzerland, 1999.

[114] *IEEE 1118, Standard Microcontroller System Serial Control Bus*. Institute of Electrical and Electronics Engineers, New York, USA, 1991.

[115] KNX Association [Official website]. http://www.knx.org/.

[116] *ISO/IEC 14543-3-1: Information technology – Home Electronic Systems (HES) Architecture – Part 3-1: Communication layers – Application layer for network based control of HES Class 1*. International Organization for Standardization, Geneve, Switzerland, 2006.

[117] Smart Energy Starts Here – Echelon. http://www.echelon.com/.

[118] LonWorks®Standards. http://www.echelon.com/technology/lonworks/standards-applications.htm.

[119] *ISO/IEC 14908-1: Information technology – Control network protocol – Part 1: Protocol stack*. International Organization for Standardization, Geneve, Switzerland, 2008.

[120] *BACnet* Website. http://www.bacnet.org/.

[121] *ISO 16484-1: Building automation and control systems – Part 1: Overview and vocabulary*. International Organization for Standardization, Geneve, Switzerland, 2010.

[122] *LCN* – Gebäudeleittechnik – Intelligente Steuerungen für Ihr Gebäude. http://www.lcn.de/.

[123] COMMERCIAL WEATHER VENDOR WEB SITES SERVING THE U.S. http://www.nws.noaa.gov/im/more.htm.

[124] Five Best Weather Web Sites. http://lifehacker.com/5897973/five-best-weather-web-sites.

[125] Comparing Weather *APIs* | Michael Welburn. http://michaelwelburn.com/2011/11/02/comparing-weather-apis/.

[126] Tim Bray, Jean Paoli, C.M. Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C Recommendation, 26 November 2008. `http://www.w3.org/TR/2008/REC-xml-20081126/`. Latest version available at `http://www.w3.org/TR/xml/`.

[127] *W3C* Document Object model. http://www.w3.org/DOM/.

[128] *SAX*. http://www.saxproject.org/.

[129] D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627 (Informational), July 2006.

[130] Maven – Json-lib::Welcome. http://json-lib.sourceforge.net/.

[131] FLEXJSON. http://flexjson.sourceforge.net/.

[132] google-gson – A Java library to convert JSON to Java objects and vice-versa. https://code.google.com/p/google-gson/.

[133] PHP: JSON – Manual. http://www.php.net/manual/en/book.json.php.

[134] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFCs 2817, 5785, 6266, 6585.

[135] Apache HttpComponents™. http://hc.apache.org/.

[136] Google – Official Blog: Spring cleaning in summer. http://googleblog.blogspot.co.at/2012/07/spring-cleaning-in-summer.html.

[137] Wetter und Klima – Deutscher Wetterdienst – Startseite. http://www.dwd.de/.

[138] *API* | Weather Underground. http://www.wunderground.com/weather/api/.

[139] RSS 2.0 Specification (RSS 2.0 at Harvard Law). http://cyber.law.harvard.edu/rss/rss.html.

[140] Remote *JSON* – *JSONP*. http://bob.ippoli.to/archives/2005/12/05/remote-json-jsonp/.

[141] Y. Shafranovich. Common Format and MIME Type for Comma-Separated Values (CSV) Files. RFC 4180 (Informational), October 2005.

[142] *SYNOP* Data Format (FM-12) – Surface Synoptic Observations. http://weather.unisys.com/wxp/Appendices/Formats/SYNOP.html.

[143] United States. *Aeronautical information manual [electronic resource] : official guide to basic flight information and ATC procedures*. Federal Aviation Administration [Washington, D.C.].

[144] INTERNATIONAL CIVIL AVIATION ORGANIZATION – A United Nations Specialized Agency. http://www.icao.int/Pages/default.aspx.

[145] PyMETAR. http://www.schwarzvogel.de/software-pymetar.shtml.

[146] matthew feldt – projects – java metar parsing utility. http://www.feldt.com/work/projects/metar/.

[147] Yahoo! Weather – Yahoo! Developer Network. http://developer.yahoo.com/weather/.

[148] World Weather Online®– Global weather forecast and weather content provider. http://www.worldweatheronline.com.

[149] api.met.no. http://api.yr.no/.

[150] *ADDS METARs*. http://www.aviationweather.gov/adds/metars/.

[151] National Digital Forecast Database *XML*/*SOAP* Service – *NOAA*'s National Weather Service. http://graphical.weather.gov/xml/.

[152] TODO (web page currently unavailable). http://www.weather.com/services/xmloap.html.

[153] NOAA – National Oceanic and Atmospheric Administration. http://www.noaa.gov/.

[154] J. Postel and J. Reynolds. File Transfer Protocol. RFC 959 (INTERNET STANDARD), October 1985. Updated by RFCs 2228, 2640, 2773, 3659, 5797.

[155] Martin Gudgin, Mark Hadley, Noah Mendelsohn, Moreau Jean-Jaques, Henrik Frystyk Nielsen, Anish Karmarkar, and Yves Lafon. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). W3C Recommendation, 27 April 2007. http://www.w3.org/TR/2007/REC-soap12-part1-20070427/. Latest version available at http://www.w3.org/TR/soap12-part1/.

[156] Roy Thomas Fielding. Architectural styles and the design of network-based software architectures, 2000.

[157] Creative Commons Attribution 3.0 Unported. http://creativecommons.org/licenses/by/3.0/.

[158] F.A.Q for api.met.no. http://api.yr.no/faq.html.

[159] Mark G Lawrence. The relationship between relative humidity and the dewpoint temperature in moist air: A simple conversion and applications. *Bulletin of the American Meteorological Society*, 86:225–233, 2005.

[160] B<Locationforecast> – Weather forecast for a specified place. http://api.yr.no/weatherapi/locationforecast/1.8/documentation.

[161] Schema to be used for presenting weather parameters for specific locations. http://api.yr.no/weatherapi/locationforecast/1.8/schema.

[162] B<Sunrise> – When does the sun rise and set for a given place. http://api.yr.no/weatherapi/sunrise/1.0/documentation.

[163] Ibrahim Reda and Afshin Andreas. Solar position algorithm for solar radiation applications. *Solar Energy*, 76(5):577 – 589, 2004.

[164] Manuel Blanco-Muriel, Diego C. Alarcón-Padilla, Teodoro López-Moratalla, and MartÍn Lara-Coira. Computing the solar vector. *Solar Energy*, 70(5):431 – 441, 2001.

[165] María Poveda, Mari Carmen Suárez-Figueroa, and Asunción Gómez-Pérez. Common pitfalls in ontology development. In *Proceedings of the Current topics in artificial intelligence, and 13th conference on Spanish association for artificial intelligence*, CAEPIA'09, pages 91–100, Berlin, Heidelberg, 2010. Springer-Verlag.

[166] Michael Gruninger and Mark S. Fox. The role of competency questions in enterprise engineering. In *PROCEEDINGS OF THE IFIP WG5.7 WORKSHOP ON BENCHMARKING - THEORY AND PRACTICE*, 1994.

[167] Antonio De Nicola, Michele Missikoff, and Roberto Navigli. A software engineering approach to ontology building. *Inf. Syst.*, 34:258–275, April 2009.

[168] Sally Chambless Mohamed Keshk. Model driven ontology: A new methodology for ontology development. November 2008.

[169] Mari-Carmen Suárez-Figueroa, Asunción Gómez-Pérez, Enrico Motta, and Aldo Gangemi. *Ontology Engineering in a Networked World*. Springer, Berlin, 2012.

[170] Guus Schreiber Bob, Bob Wielinga, and Wouter Jansweijer. The kactus view on the 'o' word. In *In IJCAI Workshop on Basic Ontological Issues in Knowledge Sharing*, pages 159–168, 1995.

[171] Bill Swartout, Ramesh Patil, Kevin Knight, and Tom Russ. Towards distributed use of large-scale ontologies. In *Proceedings of the 10th. Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, 1996.

[172] Hele-Mai Haav. A semi-automatic method to ontology design by using fca. In Václav Snásel and Radim Belohlávek, editors, *CLA*, volume 110 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2004.

[173] Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 1999.

[174] M. Mariano Fernández López. Overview of Methodologies for Building Ontologies. In *Proceedings of the IJCAI-99 Workshop on Ontologies and Problem Solving Methods (KRR5) Stockholm, Sweden, August 2, 1999*, 1999.

[175] Mariano Fernández-López. Overview and analysis of methodologies for building ontologies. *Knowl. Eng. Rev.*, 17(2):129–156, June 2002.

[176] Dean Jones, Trevor Bench-capon, and Pepijn Visser. Methodologies for ontology development. pages 62–75, 1998.

[177] Oscar Corcho, Mariano Fernández-López, and Asunción Gómez-Pérez. Methodologies, tools and languages for building ontologies: where is their meeting point? *Data Knowl. Eng.*, 46(1):41–64, July 2003.

[178] MIKE USCHOLD, MARTIN KING, STUART MORALEE, and YANNIS ZORGIOS. The enterprise ontology. *The Knowledge Engineering Review*, 13:31–89, 2 1998.

[179] Alexander Osterwalder and Yves Pigneur. An e-business model ontology for modeling e-business, 2002.

[180] Rinke Hoekstra, Joost Breuker, Marcello Di Bello, and Alexander Boer. Lkif core: Principled ontology development for the legal domain. In *Proceedings of the 2009 conference on Law, Ontologies and the Semantic Web: Channelling the Legal Information Flood*, pages 21–52, Amsterdam, The Netherlands, The Netherlands, 2009. IOS Press.

[181] Mario Gutiérrez A., Alejandra García-Rojas, Daniel Thalmann, Frederic Vexo, Laurent Moccozet, Nadia Magnenat-Thalmann, Michela Mortara, and Michela Spagnuolo. An ontology of virtual humans. *The Visual Computer*, 23(3):207–218, 2007.

[182] Nicolas Anquetil, Káthia M. de Oliveira, Kleiber D. de Sousa, and Márcio G. Batista Dias. Software maintenance seen as a knowledge management issue. *Inf. Softw. Technol.*, 49(5):515–529, May 2007.

[183] Li-Yen Shue, Ching-Wen Chen, and Weissor Shiue. The development of an ontology-based expert system for corporate financial rating. *Expert Systems with Applications*, 36(2, Part 1):2130 – 2142, 2009.

[184] Muhammad Shahab Siddiqui, Zubair A. Shaikh, and Abdul Rahman Memon. Towards the development of human community ontology. In *Proceedings of the 2009 WRI World Congress on Software Engineering - Volume 03*, WCSE '09, pages 8–12, Washington, DC, USA, 2009. IEEE Computer Society.

[185] Jeffrey Undercoffer, Anupam Joshi, and John Pinkston. Modeling computer attacks: An ontology for intrusion detection. In *In: 6th International Symposium on Recent Advances in Intrusion Detection*, pages 113–135. Springer, 2003.

[186] Joanne S Luciano. PAX of mind for pathway researchers. *Drug Discov Today*, 10(13):937–942, July 2005.

[187] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, Reading, Mass., 1999.

[188] James Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified Modeling Language Reference Manual, The (2nd Edition)*. Pearson Higher Education, 2004.

[189] Rainer Ruggaber. Athena-advanced technologies for interoperability of heterogeneous enterprise networks and their applications.

[190] M. Al-Yahya, H. Alkhalifa, A. Bahanshal, I. Alodah, and N. Al-Helwah. An ontological model for representing computational lexicons a componential based approach. In *Natural Language Processing and Knowledge Engineering (NLP-KE), 2010 International Conference on*, pages 1–6, 2010.

[191] Gregorio D'Agostino and Antonio De Nicola. Towards semantic profiling in social networks.

[192] Michele Missikoff and Francesco Taglino. A semantic cooperation and interoperability platform for the european chambers of commerce. In Tomas Vitvar, Vassilios Peristeras, and Konstantinos Tarabanis, editors, *Semantic Technologies for E-Government*, pages 129–149. Springer Berlin Heidelberg, 2010.

[193] LD-CAST Project – Home. http://www.ldcastproject.com/.

[194] O. Corcho, M. Fernández, A. Gómez-Pérez, and A. López-Cima. Building legal ontologies with methontology and webode. In R. Benjamins, P. Casanovas, J. Breuker, and A. Gangemi, editors, *Law and the Semantic Web*, number 3369 in LNAI, pages 142–157. Springer-Verlag, 2005.

[195] Mariano Fernández López, Asunción Gómez-Pérez, Juan Pazos Sierra, and Alejandro Pazos Sierra. Building a chemical ontology using methontology and the ontology design environment. *IEEE Intelligent Systems*, 14(1):37–46, January 1999.

[196] Jinsoo Park, Kimoon Sung, and Sewon Moon. Developing graduation screen ontology based on the methontology approach. *Networked Computing and Advanced Information Management, International Conference on*, 2:375–380, 2008.

[197] Zakaria Yuslina, Deris Safaai, and Othman Muhammad Razib. The development of ontology for metabolic pathways using methontology. In *Proceedings of the Postgraduate Annual Research Seminar*, pages 291–295, 2005.

[198] Zacharias Candeias Jr Fred Freitas and Heiner Stuckenschmidt. A new usage for semantic technologies for egovernment: Checking official documents' consistency. *Electronic Journal of e-Government*, 8:121–134, 2010.

[199] Andrés Iglesias-Pérez, Marino Linaje, Juan Carlos Preciado, Fernando Sánchez-Figueroa, Elena Gómez-Martínez, Rafael González-Cabero, and José Ángel Martínez-Usero. A context-aware semantic approach for the effective selection of an assistive software.

[200] Richard A Smith. Designing a cartographic ontology for use with expert systems. In *proceedings: A special joint symposium of ISPRS Technical Commission IV & AutoCarto in conjunction with ASPRS/CaGIS 2010 Fall Specialty Conference*, 2010.

[201] Marlos Silva, Endhe Elias Soares, Evandro Costa, Ig Ibert Bittencourt, Heitor Barros, Leandro Dias, Douglas Véras, and Alan P Silva. Combining methontology and a model driven ontology approach to build an educational ontology. *IEEE Technology and Engineering Education (ITEE)*, 6(3):11–18, 2011.

[202] Evandro de Barros Costa. *Um modelo de ambiente interativo de aprendizagem baseado numa arquitetura multi-agentes*. PhD thesis, PhD thesis, Universidade Federal da Paraıba, 1997.

[203] LM Vilches-Blázquez, JA Ramos, FJ López-Pellicer, O Corcho, and J Nogueras-Iso. An approach to comparing different ontologies in the context of hydrographical information. In *Information fusion and geographic information systems*, pages 193–207. Springer, 2009.

[204] Asunción Gómez-Pérez, Mariano Fernández, Facultad De Informtica, E. U. Politcnica, and Campus Universitario Ctra. Towards a method to conceptualize domain ontologies.

[205] Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez, and Boris Villazón-Terrazas. How to write and use the ontology requirements specification document. In *Proceedings of the Confederated International Conferences, CoopIS, DOA, IS, and ODBASE 2009 on On the Move to Meaningful Internet Systems: Part II*, OTM '09, pages 966–982, Berlin, Heidelberg, 2009. Springer-Verlag.

[206] B Boehm. A spiral model of software development and enhancement. *SIGSOFT Softw. Eng. Notes*, 11(4):14–24, August 1986.

[207] J. Bayer and Dirk Muthig. A view-based approach for improving software documentation practices. In *Engineering of Computer Based Systems, 2006. ECBS 2006. 13th Annual IEEE International Symposium and Workshop on*, pages 10 pp.–278, 2006.

[208] American Meteorological Society Home Page. http://www.ametsoc.org/.

[209] Pellint: An Ontology Repair Tool – Clark & Parsia: Thinking Clearly. http://weblog.clarkparsia.com/2008/07/02/pellint-an-ontology-repair-tool/.

[210] Junit. a programmer-oriented testing framework for java. http://junit.org/.

[211] D. Binkley, M. Davis, D. Lawrie, and C. Morrell. To camelcase or under_score. In *Program Comprehension, 2009. ICPC '09. IEEE 17th International Conference on*, pages 158–167, 2009.

[212] Javadoc tool home page. http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html.

[213] Cobertura. http://cobertura.sourceforge.net/.

[214] Apache Ant™. http://ant.apache.org/.

[215] Jar file specification. http://docs.oracle.com/javase/6/docs/technotes/guides/jar/jar.html.

[216] Properties (java platform se 7). http://docs.oracle.com/javase/7/docs/api/java/util/Properties.html#load(java.io.Rea

[217] Hazewinkel Michiel. *Encyclopidia of Mathematics*. Kluwer, 1994.

[218] Plataforma solar de almería - psa algorithm files. http://www.psa.es/sdg/sunpos.htm.

[219] ARK | Intel®Core™2 Quad Processor Q6600 (8M Cache, 2.40 GHz, 1066 MHz FSB). http://ark.intel.com/products/29765.

[220] The world's most popular free OS | Ubuntu. http://www.ubuntu.com/.

A Weather Ontology for Predictive Control in Smart Homes

Staroch, Paul

01 **Mario J. Kofler** Page no. **9**

7/8/2013 8:43

support

02 **Mario J. Kofler** Page no. **9**

7/8/2013 8:43

while

03 **Mario J. Kofler** Page no. **11**

7/8/2013 8:44

... zu senken.

04 **Mario J. Kofler** Page no. **11**

7/8/2013 8:44

eines Projekts

05 **Mario J. Kofler** Page no. **11**

7/8/2013 8:45

Durchgehend: Smart Home sollte nicht kursiv gestellt werden, nur an der Stelle an welcher Smart Homes definiert werden.

06 **Mario J. Kofler** Page no. **11**

7/8/2013 8:46

Systemen

07 **Mario J. Kofler** Page no. **11**

7/8/2013 8:48

besser: Smart Homes

08 **Mario J. Kofler** Page no. **17**

7/8/2013 8:49

besser: support

09 **Mario J. Kofler** Page no. **17**

7/8/2013 8:51

besser: may manage

10 **Mario J. Kofler** Page no. **17**

7/8/2013 8:52

streichen

11 **Mario J. Kofler** Page no. **17**

7/8/2013 8:51

besser: adjust

12 **Mario J. Kofler** Page no. **17**

7/8/2013 8:53

increasing

13 **Mario J. Kofler** Page no. **18**

7/8/2013 8:54

streichen

14 **Mario J. Kofler** Page no. **18**

7/8/2013 8:54

evolution

15 **Mario J. Kofler** Page no. **18**

7/8/2013 8:55

besser: "automated home"

16 **Mario J. Kofler** Page no. **18**

7/8/2013 8:55

are often

17 **Mario J. Kofler** Page no. **19**

7/8/2013 8:56

streichen

18 **Mario J. Kofler** Page no. **19**

7/8/2013 8:57

source

19 **Mario J. Kofler** Page no. **19**

7/8/2013 8:57

besser: needs

20 **Mario J. Kofler** Page no. **19**

7/8/2013 9:00

may be required by

21 **Mario J. Kofler** Page no. **19**

7/8/2013 9:01

covers

22 **Mario J. Kofler** Page no. **19**

7/8/2013 9:01

future weather states

23 **Mario J. Kofler** Page no. **19**

7/8/2013 9:02

streichen

24 **Mario J. Kofler** Page no. **19**

7/8/2013 9:02

one particular

25 **Mario J. Kofler** Page no. **20**

7/8/2013 9:03

usage in

26 **Mario J. Kofler** Page no. **20**

7/8/2013 9:04

in

27 **Mario J. Kofler** Page no. **20**

7/8/2013 9:05

leerzeichen fehlt

28 **Mario J. Kofler** Page no. **21**

7/8/2013 9:05

weather data parameters

29 **Mario J. Kofler** Page no. **23**

7/8/2013 9:06

ontologies and associated modeling languages such as RDF, RDFS and OWL

30 **Mario J. Kofler** Page no. **23**

7/8/2013 9:06

in the SmartHomeWeather ontology

31 **Mario J. Kofler** Page no. **23**

7/8/2013 9:07

utilizing

32 **Mario J. Kofler** Page no. **24**

7/8/2013 9:07

related to it

33 **Mario J. Kofler** Page no. **24**

7/8/2013 9:07

deduce

34 **Mario J. Kofler** Page no. **25**

7/8/2013 9:08

The Semantic Web

35 **Mario J. Kofler** Page no. **25**

7/8/2013 9:08

besser: machine-interpretable

36 **Mario J. Kofler** Page no. **26**

7/8/2013 9:10

"individuals" werden als solches erst in OWL eingeführt. "data values", "instances" etc. wäre hier besser.

37 **Mario J. Kofler** Page no. **26**

7/8/2013 9:10

in

38 **Mario J. Kofler** Page no. **28**

7/8/2013 13:08

Referenzen auf Tables, Chapters, Figures oder Sections sollten immer mit großem Anfangsbuchstaben gekennzeichnet werden.

39 **Mario J. Kofler** Page no. **28**

7/8/2013 9:13

hier wohl eher "with domain X and range Y" oder?

40 **Mario J. Kofler** Page no. **29**

7/8/2013 9:13

property

41 **Mario J. Kofler** Page no. **29**

7/8/2013 9:13

besser: the defined

42 **Mario J. Kofler** Page no. **29**

7/8/2013 9:16

vielleicht sollte man hier kurz die Non-Unique-Names Assumption erwähnen damit klar wird warum diese Statements überhaupt notwendig sind

43 **Mario J. Kofler** Page no. **29**

7/8/2013 9:18

... if no other classes exist.

44 **Mario J. Kofler** Page no. **29**

7/8/2013 9:26

or classes

45 **Mario J. Kofler** Page no. **30**

7/8/2013 9:27

unklar: entweder ausformulieren oder streichen.

46 **Mario J. Kofler** Page no. **30**

7/8/2013 9:28

Gehört dieses Kapitel wirklich zu den "Foundations"?

47 **Mario J. Kofler** Page no. **30**

7/8/2013 9:28

streichen

48 **Mario J. Kofler** Page no. **30**

7/8/2013 9:29

streichen

49 **Mario J. Kofler** Page no. **31**

7/8/2013 9:29

streichen

50 **Mario J. Kofler** Page no. **31**

7/8/2013 9:29

streichen

51 **Mario J. Kofler** Page no. **31**

7/8/2013 9:30

To investigate the possibility if...

52 **Mario J. Kofler** Page no. **31**

7/8/2013 9:31

streichen

53 **Mario J. Kofler** Page no. **32**

7/8/2013 9:31

yields to

54 **Mario J. Kofler** Page no. **33**

7/8/2013 9:32

The

55 **Mario J. Kofler** Page no. **34**

7/8/2013 9:33

streichen

56 **Mario J. Kofler** Page no. **34**

7/8/2013 9:33

streichen

57 **Mario J. Kofler** Page no. **34**

7/8/2013 9:34

TYPO

58 **Mario J. Kofler** Page no. **35**

7/8/2013 9:35

...already explained in the case of...

59 **Mario J. Kofler** Page no. **38**

7/8/2013 9:35

probably remain

60 **Mario J. Kofler** Page no. **38**

7/8/2013 9:37

of the property

61 **Mario J. Kofler** Page no. **39**

7/8/2013 9:38

ontology merely impossible.

62 **Mario J. Kofler** Page no. **39**

7/8/2013 9:38

through

63 **Mario J. Kofler** Page no. **40**

7/8/2013 9:39

include concepts and properties that are written/defined to represent

64 **Mario J. Kofler** Page no. **41**

7/8/2013 9:40

useful data on exterior influences

65 **Mario J. Kofler** Page no. **41**

7/8/2013 9:39

streichen

66 **Mario J. Kofler** Page no. **41**

7/8/2013 9:39

weather data

67 **Mario J. Kofler** Page no. **41**

7/8/2013 9:40

are the

68 **Mario J. Kofler** Page no. **42**

7/8/2013 9:41

besser: the system

69 **Mario J. Kofler** Page no. **42**

7/8/2013 9:41

providing a

70 **Mario J. Kofler** Page no. **42**

7/8/2013 9:42

besser: "...a one-world description of the current weather situation."

71 **Mario J. Kofler** Page no. **42**

7/8/2013 9:42

besser: general weather conditions

72 **Mario J. Kofler** Page no. **42**

7/8/2013 9:44

hier sollten konkret ein oder zwei weather elements genannt werden auf die diese Aussage zutrifft.

73 **Mario J. Kofler** Page no. **43**

7/8/2013 9:45

...is in this section used as an example to picture/illustrate...

74 **Mario J. Kofler** Page no. **44**

7/8/2013 9:46

streichen

75 **Mario J. Kofler** Page no. **44**

7/8/2013 9:46

exist

76 **Mario J. Kofler** Page no. **44**

7/8/2013 9:47

vielleicht auch "air pollution"?

77 **Mario J. Kofler** Page no. **44**

7/8/2013 9:47

besser: presents

78 **Mario J. Kofler** Page no. **44**

7/8/2013 9:47

usage

79 **Mario J. Kofler** Page no. **45**

7/8/2013 9:48

Klammer zu.

80 **Mario J. Kofler** Page no. **45**

7/8/2013 9:48

the more suitable a weather service is

81 **Mario J. Kofler** Page no. **45**

7/8/2013 9:50

aus welchem Grund 48 Stunden, und nicht 24, wenn nur max. 24-Stunden-Vorhersagen verwendet werden?

82 **Mario J. Kofler** Page no. **46**

7/8/2013 9:58

streichen

83 **Mario J. Kofler** Page no. **46**

7/8/2013 9:58

streichen

84 **Mario J. Kofler** Page no. **46**

7/8/2013 9:59

sensoren könnten andere elemente zur Verfügung stellen, welche nicht in Wetter Services vorkommen (siehe p. 34)

85 **Mario J. Kofler** Page no. **46**

7/8/2013 10:00

Tabelle 3.1 beschreibt etwas anderes.

86 **Mario J. Kofler** Page no. **46**

7/8/2013 10:01

updates

87 **Mario J. Kofler** Page no. **47**

7/8/2013 10:02

provides

88 **Mario J. Kofler** Page no. **47**

7/8/2013 10:01

streichen

89 **Mario J. Kofler** Page no. **47**

7/8/2013 10:05

issue

90 **Mario J. Kofler** Page no. **48**

7/8/2013 10:07

Conclusion kommt an diesem Punkt sehr plötzlich und mitten im Kapitel. ich würde diese Section aufteilen: Teile davon gehen hier als Introduction für Section 3.5 und 3.6 ein. Andere Teile werden für eine Conclusion am Ende des Kapitels (nach 3.6) verwendet.

91 **Mario J. Kofler** Page no. **48**

7/8/2013 10:07

elements specified in the ontology,

92 **Mario J. Kofler** Page no. **48**

7/8/2013 10:07

besser: following

93 **Mario J. Kofler** Page no. **48**

7/8/2013 10:08

streichen

94 **Mario J. Kofler** Page no. **49**

7/8/2013 10:09

usage

95 **Mario J. Kofler** Page no. **49**

7/8/2013 10:09

streichen

96 **Mario J. Kofler** Page no. **49**

7/8/2013 10:09

usage by

97 **Mario J. Kofler** Page no. **49**

7/8/2013 10:09

of charge

98 **Mario J. Kofler** Page no. **49**

7/8/2013 10:10

Hier reicht es zu sagen, dass die Premium API kostenpflichtig ist.

99 **Mario J. Kofler** Page no. **50**

7/8/2013 10:10

streichen

100 **Mario J. Kofler** Page no. **50**

7/8/2013 10:10

umformulieren

101 **Mario J. Kofler** Page no. **50**

7/8/2013 13:17

typo

102 **Mario J. Kofler** Page no. **50**

7/8/2013 10:11

clash mit p.30 und Beschreibung, dass nur Elemente übernommen werden, welche in Wetter Services vorhanden sind.

103 **Mario J. Kofler** Page no. **50**

7/8/2013 10:11

value

104 **Mario J. Kofler** Page no. **52**

7/8/2013 10:12

a

105 **Mario J. Kofler** Page no. **52**

7/8/2013 10:12

falsche Referenz

106 **Mario J. Kofler** Page no. **52**

7/8/2013 10:12

of Norway

107 **Mario J. Kofler** Page no. **53**

7/8/2013 10:13

typo

108 **Mario J. Kofler** Page no. **53**

7/8/2013 10:13

typo

109 **Mario J. Kofler** Page no. **53**

7/8/2013 10:13

alignment

110 **Mario J. Kofler** Page no. **54**

8/7/2013 14:49

elements that can be retrieved from yr.no

111 **Mario J. Kofler** Page no. **54**

7/8/2013 10:15

Hier könnte noch etwas mehr darüber gesagt werden.

112 **Mario J. Kofler** Page no. **55**

7/8/2013 10:15

Was ist der Grund dafür?

113 **Mario J. Kofler** Page no. **55**

7/8/2013 10:16

3.7 Conclusion ....

114 **Mario J. Kofler** Page no. **57**

7/8/2013 10:18

later

115 **Mario J. Kofler** Page no. **59**

7/8/2013 10:19

One methodology

116 **Mario J. Kofler** Page no. **59**

7/8/2013 10:19

may delegate

117 **Mario J. Kofler** Page no. **59**

7/8/2013 10:19

streichen

118 **Mario J. Kofler** Page no. **59**

7/8/2013 10:19

streichen

119 **Mario J. Kofler** Page no. **59**

7/8/2013 10:20

streichen

120 **Mario J. Kofler** Page no. **60**

7/8/2013 10:22

besser: gathered

121 **Mario J. Kofler** Page no. **60**

7/8/2013 10:22

besser: explain

122 **Mario J. Kofler** Page no. **61**

7/8/2013 10:22

besser: go

123 **Mario J. Kofler** Page no. **61**

7/8/2013 10:27

how to

124 **Mario J. Kofler** Page no. **62**

7/8/2013 10:30

hier fehlt was.

125 **Mario J. Kofler** Page no. **62**

7/8/2013 10:30

hier fehlt was.

126 **Mario J. Kofler** Page no. **63**

7/8/2013 10:31

iterates over

127 **Mario J. Kofler** Page no. **63**

7/8/2013 10:31

of this

128 **Mario J. Kofler** Page no. **63**

7/8/2013 10:32

besser: advise

129 **Mario J. Kofler** Page no. **63**

7/8/2013 10:32

streichen

130 **Mario J. Kofler** Page no. **63**

7/8/2013 10:32

it also comes

131 **Mario J. Kofler** Page no. **65**

7/8/2013 10:33

laut dieser Beschreibung gehören die Phasen "Iterations" und "Phases" in Figure 4.4 vertauscht

132 **Mario J. Kofler** Page no. **65**

7/8/2013 10:34

structuring

133 **Mario J. Kofler** Page no. **65**

7/8/2013 10:34

analyses

134 **Mario J. Kofler** Page no. **65**

7/8/2013 10:34

besser domain:

135 **Mario J. Kofler** Page no. **66**

7/8/2013 10:35

generating

136 **Mario J. Kofler** Page no. **66**

7/8/2013 10:35

Klammer fehlt.

137 **Mario J. Kofler** Page no. **66**

7/8/2013 10:36

besser: ... to find actual projects using it...

138 **Mario J. Kofler** Page no. **67**

7/8/2013 10:37

Die Verwendung der Kursivschrift ist hier etwas irreführend und sollte überarbeitet werden

139 **Mario J. Kofler** Page no. **67**

7/8/2013 10:39

document

140 **Mario J. Kofler** Page no. **67**

7/8/2013 10:40

these ... documents?

141 **Mario J. Kofler** Page no. **67**

7/8/2013 10:40

Ich würde wie bereits bei den anderen beschriebenen Methodologien den Überblicksgraph (Fig. 4.6) bereits hier einfügen und danach immer auf diese Figure referenzieren -> gibt ein einheitlicheres Bild.

142 **Mario J. Kofler** Page no. **68**

7/8/2013 10:41

streichen

143 **Mario J. Kofler** Page no. **68**

7/8/2013 10:42

Ebenso wie beim letzten Kapitel würde ich hier wiederum die Conclusion aufteilen: Einerseits würde ich ein Kapitel 4.3 Summary machen und andererseits wieder eine Conclusion am Ende des Kapitels.

144 **Mario J. Kofler** Page no. **68**

7/8/2013 10:42

methodologies for

145 **Mario J. Kofler** Page no. **68**

7/8/2013 10:42

gave

146 **Mario J. Kofler** Page no. **68**

7/8/2013 10:43

streichen

147 **Mario J. Kofler** Page no. **68**

7/8/2013 10:43

is acceptable

148 **Mario J. Kofler** Page no. **69**

7/8/2013 10:44

towards a

149 **Mario J. Kofler** Page no. **69**

7/8/2013 10:44

defined

150 **Mario J. Kofler** Page no. **69**

7/8/2013 10:44

for

151 **Mario J. Kofler** Page no. **69**

7/8/2013 10:46

umschreiben: "... that does not exactly match the methodology presented in any of the papers and combines aspects from several papers."

152 **Mario J. Kofler** Page no. **69**

7/8/2013 10:46

besser: the process

153 **Mario J. Kofler** Page no. **69**

7/8/2013 10:47

end-users of the planned ontology

154 **Mario J. Kofler** Page no. **69**

7/8/2013 10:47

are to be integrated

155 **Mario J. Kofler** Page no. **70**

7/8/2013 10:48

besser: meets the previously specified requirements.

156 **Mario J. Kofler** Page no. **70**

7/8/2013 10:48

which the ontology moves through...

157 **Mario J. Kofler** Page no. **70**

7/8/2013 10:49

anderer Titel, vielleicht "METHONTOLOGY approach"?

158 **Mario J. Kofler** Page no. **71**

7/8/2013 10:51

involving

159 **Mario J. Kofler** Page no. **71**

7/8/2013 10:52

presents a template for the respective artefact.

160 **Mario J. Kofler** Page no. **71**

7/8/2013 10:52

specifies

161 **Mario J. Kofler** Page no. **71**

7/8/2013 10:53

streichen

162 **Mario J. Kofler** Page no. **71**

7/8/2013 10:53

heavily depend

163 **Mario J. Kofler** Page no. **71**

7/8/2013 10:53

evolutionary

164 **Mario J. Kofler** Page no. **71**

7/8/2013 10:54

are

165 **Mario J. Kofler** Page no. **73**

7/8/2013 10:54

tree.

166 **Mario J. Kofler** Page no. **74**

7/8/2013 10:54

These

167 **Mario J. Kofler** Page no. **74**

7/8/2013 10:55

taxonomies.

168 **Mario J. Kofler** Page no. **75**

7/8/2013 10:55

Each

169 **Mario J. Kofler** Page no. **75**

7/8/2013 10:56

these axioms must be described

170 **Mario J. Kofler** Page no. **76**

7/8/2013 10:56

a rules table must be built

171 **Mario J. Kofler** Page no. **76**

7/8/2013 10:57

...then <consequent>: hereby, <conditions>...

172 **Mario J. Kofler** Page no. **76**

7/8/2013 10:57

of

173 **Mario J. Kofler** Page no. **76**

7/8/2013 10:57

performed or described

174 **Mario J. Kofler** Page no. **76**

7/8/2013 10:58

streichen

175 **Mario J. Kofler** Page no. **77**

7/8/2013 10:58

[108]:

176 **Mario J. Kofler** Page no. **77**

7/8/2013 10:59

gathered during the process described in

177 **Mario J. Kofler** Page no. **77**

7/8/2013 10:59

initially specified (see previous paragraphs)

178 **Mario J. Kofler** Page no. **77**

7/8/2013 11:00

of

179 **Mario J. Kofler** Page no. **77**

7/8/2013 11:00

must hereby take

180 **Mario J. Kofler** Page no. **77**

7/8/2013 11:00

or generally

181 **Mario J. Kofler** Page no. **78**

7/8/2013 11:01

streichen

182 **Mario J. Kofler** Page no. **78**

7/8/2013 11:01

streichen

183 **Mario J. Kofler** Page no. **78**

7/8/2013 11:01

occurred

184 **Mario J. Kofler** Page no. **78**

7/8/2013 11:02

are also again

185 **Mario J. Kofler** Page no. **78**

7/8/2013 11:02

4.5 Conclusion ....

186 **Mario J. Kofler** Page no. **80**

7/8/2013 11:42

streichen.

187 **Mario J. Kofler** Page no. **81**

7/8/2013 11:45

ist der end-user nicht das Smart Home System selbst?

188 **Mario J. Kofler** Page no. **81**

7/8/2013 11:45

besser: weather state

189 **Mario J. Kofler** Page no. **82**

7/8/2013 11:46

streichen

190 **Mario J. Kofler** Page no. **83**

7/8/2013 11:46

(cf. ...)

191 **Mario J. Kofler** Page no. **84**

7/8/2013 11:47

streichen

192 **Mario J. Kofler** Page no. **84**

7/8/2013 11:47

streichen

193 **Mario J. Kofler** Page no. **84**

7/8/2013 11:49

"Cold"?

194 **Mario J. Kofler** Page no. **84**

7/8/2013 11:49

streichen

195 **Mario J. Kofler** Page no. **86**

7/8/2013 11:49

five

196 **Mario J. Kofler** Page no. **86**

7/8/2013 11:50

besser: consists of only one element

197 **Mario J. Kofler** Page no. **86**

7/8/2013 11:51

Definition von "okta"

198 **Mario J. Kofler** Page no. **88**

7/8/2013 11:52

typo

199 **Mario J. Kofler** Page no. **93**

7/8/2013 11:59

Diese Grafik zeigt einen sehr zentralen Output der Diplomarbeit. Diese Grafik gehört daher noch etwas genauer erklärt und argumentiert warum die Hierarchie genau so definiert wurde. Es ist auch nicht klar ob hier vielleicht einzelne Konzepte und Abhängigkeiten noch anders definert werden sollten: Warum ist beispielsweise "severe weather" eine subklasse von "NoAwningWeather", "Very rainy weather" und "Stormy weather"? sollte es nicht umgekehrt sein? Sind nicht die 3 genannten im Allgemeinen eher Spezialisierungen einer "Severe Weather" Klasse, und "Thunderstorm" eine Spezialisierung/Subklasse von "Stormy Weather"? Solche Überlegungen könnte man auch in Bezug auf "Fair Weather" anstellen. Diese Beziehungen sollten nochmal gründlich überlegt werden.

200 **Mario J. Kofler** Page no. **94**

7/8/2013 12:00

usage of

201 **Mario J. Kofler** Page no. **95**

7/8/2013 12:02

Im Rahmen der Implementation ist es unter Anderem auch wichtig das Reasoning zumindest kurz anzureissen. Dies kann zum Beispiel in einer eigenen Subsection passieren. Hierbei könntest du unter anderem das Reasoning in Bezug auf Figure 5.14 etwas näher ausführen.

202 **Mario J. Kofler** Page no. **96**

7/8/2013 12:05

besser: vielleicht "temporal entities"?

203 **Mario J. Kofler** Page no. **96**

7/8/2013 12:07

An diesem Punkt nochmals die "geo" Ontologie erwähnen

204 **Mario J. Kofler** Page no. **98**

7/8/2013 12:07

streichen

205 **Mario J. Kofler** Page no. **98**

7/8/2013 12:08

streichen

206 **Mario J. Kofler** Page no. **100**

7/8/2013 12:13

Es wäre wünschenswert (vielleicht in einer eigenen Subsection) zumindest ein Beispiel einer SPARQL query sowie eine SWRL Rule zu zeigen, welche darlegen wie mit Hilfe dieser Abfragen noch weitere Competency Questions beantwortet werden können. Daher ist hier die Erweiterung des Kapitels um eine weitere Subsection vorzunehmen

207 **Mario J. Kofler** Page no. **101**

15/7/2013 13:37

besser: discusses

208 **Mario J. Kofler** Page no. **101**

7/8/2013 12:18

besser: of

209 **Mario J. Kofler** Page no. **101**

7/8/2013 12:19

allow a simple

210 **Mario J. Kofler** Page no. **102**

7/8/2013 12:20

alignment

211 **Mario J. Kofler** Page no. **104**

7/8/2013 12:20

Auf welche Art und Weise wird die ontologie modifiziert?

212 **Mario J. Kofler** Page no. **105**

7/8/2013 12:21

alignment

213 **Mario J. Kofler** Page no. **106**

7/8/2013 12:22

Hierbei wäre eine graphische Anschauung des Workflows sehr hilfreich.

214 **Mario J. Kofler** Page no. **106**

7/8/2013 12:22

in

215 **Mario J. Kofler** Page no. **106**

7/8/2013 12:23

hier fehlt etwas....

216 **Mario J. Kofler** Page no. **110**

7/8/2013 12:25

umformulieren

217 **Mario J. Kofler** Page no. **110**

7/8/2013 12:30

dieser Satz ist unklar...

218 **Mario J. Kofler** Page no. **111**

7/8/2013 12:34

Ich finde die Conclusion gehört nochmals überarbeitet. Hierbei sollten nicht nur die Chapters sowie ihr Inhalt nochmals präsentiert werden, sondern vielmehr ein Resümee gezogen werden, was mit dieser Diplomarbeit erreicht wurde, sowie nochmals der "main outcome" in Kurzfassung beschrieben werden. Der Inhaltsüberblick über die einzelnen Chapter kann gerne drin bleiben, jedoch muss die Conclusion um die genannte Zusammenfassung erweitert werden.

219 **Mario J. Kofler** Page no. **111**

7/8/2013 12:36

besser: This thesis presents...

220 **Mario J. Kofler** Page no. **111**

7/8/2013 12:36

discusses

221 **Mario J. Kofler** Page no. **112**

7/8/2013 12:37

besser: shortcomings

222 **Mario J. Kofler** Page no. **112**

7/8/2013 12:38

besser: shortcomings

223 **Mario J. Kofler** Page no. **112**

7/8/2013 12:38

streichen

224 **Mario J. Kofler** Page no. **112**

7/8/2013 12:38

vielleicht eine bessere Formulierung finden

225 **Mario J. Kofler** Page no. **112**

7/8/2013 12:38

these

226 **Mario J. Kofler** Page no. **112**

7/8/2013 12:40

... it is certain that it increases ....

227 **Mario J. Kofler** Page no. **113**

7/8/2013 12:40

at the present time, it...

228 **Mario J. Kofler** Page no. **113**

7/8/2013 12:50

Hier gehört meiner Meinung nach kein Absatz.

229 **Mario J. Kofler** Page no. **113**

7/8/2013 12:41

streichen