Applied Artificial Intelligence – An Engineering Approach

Book · March 2016

CITATIONS
3

READS
6,690

1 author:

Bernhard G. Humm
Darmstadt University of Applied Sciences
95 PUBLICATIONS **520** CITATIONS

Bernhard G. Humm

Applied Artificial Intelligence

An Engineering Approach

Applied Artificial Intelligence

An Engineering Approach

Bernhard G. Humm

This book is for sale at http://leanpub.com/AAI

This version was published on 2016-06-11

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Contents

Preface

Why yet another book on Artificial Intelligence?

It is true that hundreds of publications on Artificial Intelligence (AI) have been published within the last decades - scientific papers and text books. Most of them focus on the theory behind AI solutions: logic, reasoning, statistical foundations, etc. However, little can be found on engineering AI applications.

Modern, complex IT applications are not built from scratch but by integrating off-the-shelf components: libraries, frameworks, and services. The same applies, of course, for AI applications. Over the last decades, numerous off-the-shelf components for AI base functionality such as logic, reasoning, and statistics have been implemented - commercial and open source. Integrating such components into user friendly, high-performance, and maintainable AI applications requires specific engineering skills. This book focuses on those skills.

My own professional background is that of a software engineer. After under-/postgraduate studies of Computer Science in Germany and a Ph.D. program in Australia, I worked for more than a decade in a large German software company. In extensive teams, we developed custom software for clients: multinational banks, credit-card issuers, tour operators, telecommunication providers, fashion companies and ATC authorities. My tasks were as diverse as the industry sectors and technologies used. They ranged from development, software architecture, project management to managing a division and running the company's research department.

After re-entering university as a professor some 10 years ago, a common theme of my courses has been the professional development of software according to engineering principles and practices. AI was an old love of mine but my industry projects hat little relation to AI. Noticing that AI applications - powerful image processing, speech analysis and generation etc. - were rapidly entering the market attracted my interest, again. Via R&D projects, funded by industry and the public, I gradually built up expertise in engineering AI applications. In teams with colleagues, postgraduate students and industry partners we developed applications for a hotel portal, a library, and an art museum. Currently we are developing applications for the cancer department of a hospital and a robot manufacturer. However diverse the industry sectors, many approaches and technologies can be used across the projects in order to build maintainable AI applications with a good user experience that meet functional and non-functional requirements, particularly high performance requirements. Overall, we are combining general software engineering skills with AI expertise.

Since I see a growing demand for AI applications in the business and consumer markets, I started a new M.Sc. course "Applied Artificial Intelligence" at Darmstadt University of Applied Sciences. This book reflects topics of this course. I am constantly learning: from project experience, from my colleagues and partners, from my students, and hopefully also from you, the readers of this book.

So please, don't hesitate to contact me under bernhard.humm@h-da.de when you agree or disagree with my findings.

To support you to come to grips with the topics I have added questions and exercises in all chapters. They are indicated by a pencil symbol.

Finally, I would like to thank my friend and project partner Dr Paul Walsh from NSilico Lifescience Ltd., Ireland, for valuable comments and hints.

Bernhard Humm, Darmstadt, Germany, 2016

Copyright Notice

All photos in figures of this book are published under the Creative Commons (CC) License.

- Cover image: CC Wikimedia Commons, Author Meritxell.canela, Bag_of_words.JPG[1]
- Fig. 1.2: CC Wikimedia Commons, Author Artaxerxes, 28 June 2008, Our_Community_Place_-Sandbox.jpg[2]
- Fig. 1.3: CC-BY Wikimedia Commons, Author James the photographer, 14 June 2007, Deep_-Blue.jpg[3]
- Fig. 1.4: CC Wikimedia Commons, Author Raysonho @ Open Grid Scheduler / Grid Engine, 7 April 2011, IBMWatson.jpg[4]
- Fig. 1.5: CC Wikimedia Commons, Author Michael Shick, 21 October 2015, Google_self_-driving_car_at_the_Googleplex.jpg[5]
- Fig. 1.6: Wikimedia Commons, Painter Michelangelo Buonarroti (1457-1564) {PD-Italy}{PD-Canada}, File:The_Creation_of_Adam.jpg[6]
- Fig. 2.9: CC Wikimedia Commons, Author Max Schmachtenberg, Source lod-cloud.net[7], Lod-cloud_colored_1000px.png[8]
- Fig. 6.6: CC Wikimedia Commons, Author Stephen Milborrow, 1 March 2011, CART_tree_-titanic_survivors.png[9]
- Fig. 6.19: CC Wikimedia Commons, Author Ghiles, 11 March 2016, Overfitted_Data.png[10]
- Fig. 7.2: CC Wikimedia Commons, Author Mbroemme5783, 4 October 2012, Face_Capture.PNG[11]
- Fig. 7.3: CC Wikimedia Commons, Author R Walters, 22 July 2007, Restoration.jpg[12]
- Fig. 7.4: CC Wikimedia Commons, Author OpenStax College, Source cnx.org[13], 5 April 2013, 113abcd_Medical_Imaging_Techniques.jpg[14]

[1] https://commons.wikimedia.org/wiki/File:Bag_of_words.JPG
[2] https://commons.wikimedia.org/wiki/File:Our_Community_Place_Sandbox.jpg
[3] https://commons.wikimedia.org/wiki/File:Deep_Blue.jpg
[4] https://commons.wikimedia.org/wiki/File:IBMWatson.jpg
[5] https://commons.wikimedia.org/wiki/File:Google_self_driving_car_at_the_Googleplex.jpg
[6] https://commons.wikimedia.org/wiki/File:The_Creation_of_Adam.jpg
[7] http://lod-cloud.net/
[8] https://commons.wikimedia.org/wiki/File:Lod-cloud_colored_1000px.png
[9] https://commons.wikimedia.org/wiki/File:CART_tree_titanic_survivors.png
[10] https://commons.wikimedia.org/wiki/File:Overfitted_Data.png
[11] https://commons.wikimedia.org/wiki/File:Face_Capture.PNG
[12] https://commons.wikimedia.org/wiki/File:Restoration.jpg
[13] http://cnx.org/content/col11496/1.6/
[14] https://commons.wikimedia.org/wiki/File:113abcd_Medical_Imaging_Techniques.jpg

- Fig. 7.5: Wikimedia Commons, Author KUKA Roboter GmbH, Bachmann; Source KUKA Roboter GmbH, Zugspitzstr. 140, D-86165 Augsburg, Germany, Dep. Marketing, www.kuka-robotics.com; 2003, KUKA_robot_for_flat_glas_handling.jpg[15]
- Fig. 7.6: CC Wikimedia Commons, Author Robert Basic, Source Audi A8 2013 Uploaded by AVIA BavARia, 17 October 2013, Audi_A8_2013_(11209949525).jpg[16]
- Fig. 7.7: Wikimedia Commons, Author NASA/JPL/Cornell University, Maas Digital LLC, Source photojournal.jpl.nasa.gov[17], February 2003, NASA_Mars_Rover.jpg[18]
- Fig. 7.8: CC Wikimedia Commons, Author Vazquez88, 5 February 2013, Motion_Capture_-with_Chad_Phantom.png[19]

[15] https://commons.wikimedia.org/wiki/File:KUKA_robot_for_flat_glas_handling.jpg
[16] https://commons.wikimedia.org/wiki/File:Audi_A8_2013.jpg
[17] http://photojournal.jpl.nasa.gov/catalog/PIA04413
[18] https://commons.wikimedia.org/wiki/File:NASA_Mars_Rover.jpg
[19] https://commons.wikimedia.org/wiki/File:Motion_Capture_with_Chad_Phantom.png

1. Introduction

How relevant is *Artificial Intelligence (AI)* today?

Are super-intelligent machines soon taking over the world? Or, in contrast, is AI nothing more than a burst bubble of a 20th century hype? I think neither the one nor the other is true. In my opinion, AI is *relevant and ubiquitous* in today's IT applications of business and consumer markets (although often not under the term AI).

A few examples of AI in everyday use are:

- Speech control for smart-phones, navigation systems in cars, etc.
- Face recognition in cameras
- Learning spam filters in e-mail clients
- AI in computer games
- Semantic Internet search, including question answering; See Fig. 1.1.

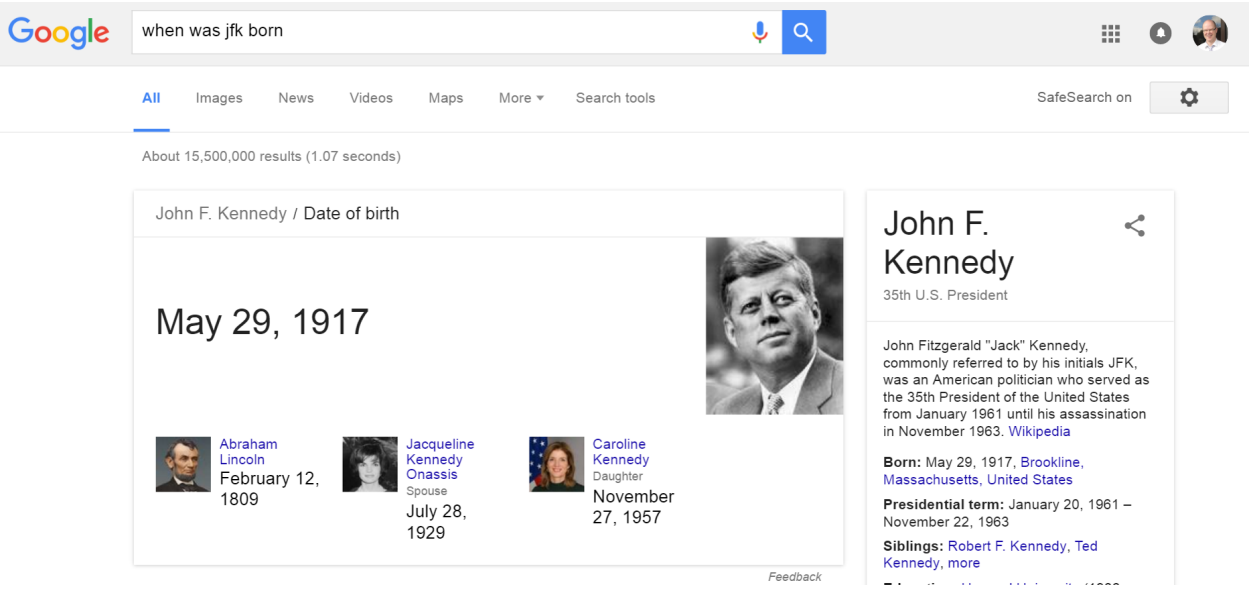

Fig. 1.1: AI in everyday use: Google's question answering "When was JFK born?"

Commercial examples are:

- Business intelligence
- Sentiment analysis

- Robotics
- Industrial computer vision
- Self-driving cars, drones, rockets (military and commercial)

How are those AI applications developed?

While most AI publications, such as scientific papers and text books, focus on the theory behind AI solutions, little can be found on engineering AI applications. What kinds of AI libraries, frameworks and services already exist? Which ones should be chosen in which situation? How to integrate them into maintainable AI applications with a good user experience? How to meet functional and non-functional requirements, in particular high performance?

The focus of this book is to answer those kinds of questions for software developers and architects.

1.1 Overview of this Book

The remainder of this chapter presents a definition of AI as well as a brief AI history. Chapter 2 presents knowledge representation mechanisms. Chapter 3 gives guidelines for the architecture of AI applications. The remaining chapters focus on individual AI areas: (4) information retrieval, (5) natural language processing, (6) machine learning, and (7) computer vision. Chapter 8 concludes the book.

Inspired by one of my AI projects, I use application examples from the domain of art museums throughout the book. Additionally, I present real-world examples from other domains.

At the end of each chapter I repeat the main themes in form of questions that you, the reader, can use as a *quick check*.

The appendix contains product tables as well as source code examples that can be used as starting point for your own developments.

1.2 What is AI?

My definition of AI (similar to many definitions in other textbooks) is:

Machines or software applications that exhibit behavior of human intelligence.

Behavior of human intelligence includes:

- *Sensing*: seeing, listening, feeling, etc.
- *Reasoning*: thinking, understanding, learning, planning etc.
- *Communicating*: speaking, writing, etc.
- *Acting*

See Fig. 1.2.

Fig. 1.2: Behaviour of human intelligence

Different *areas of AI* focus on different intelligent behavior:

- *Computer vision*: Seeing, understanding
- *Natural language processing*: Listening, speaking, reading, writing
- *Knowledge representation, reasoning, and information retrieval*: Understanding, reminding, thinking
- *Machine learning*: Learning
- *Agent technology and robotics*: All above including planning and acting

1.3 A Brief History of AI

The term "Artificial Intelligence" was coined at the *Dartmouth Workshop* in 1956. Members of this workshop were John McCarthy, Marvin Minsky, Claude Shannon, Allen Newell, Herbert Simon, and others. However, Alan Turing (1912 – 1954) with his fundamental work on computability and the so-called *Turing Test* for assessing intelligent behavior had already laid ground for AI decades before.

The 1960s - 1980s saw unprecedented *AI hype*, triggered by enthusiastic promises of quick AI results. Examples of such promises are:

"Within a generation the problem of creating 'artificial intelligence' will be substantially solved." (Marvin Minsky, 1967)

or

"Once the computers got control, we might never get it back. We would survive at their sufferance. If we're lucky, they might decide to keep us as pets." (Marvin Minsky, 1970).

This hype resulted in massive funding of AI projects, particularly in the US.

The effect of those wildly exaggerated promises was the same as in all hype. When people started to notice that the most sophisticated AI applications failed to perform tasks that are easily accomplished by small children (e.g., distinguishing an open drain cover from a shadow) they threw the baby out with the bathwater. The disillusionment of the unrealizable expectations resulted in massive cuts in funding and a collapse of the AI market. The 1980s - 1990s are sometimes called the *AI winter*.

From then on, unnoticed and often not under the term AI, AI methods and technologies have matured enormously driven by major technology companies. For example, Google co-founder Larry Page[1] said in 2006:

“We want to create the ultimate search engine that can understand anything. Some people could call that artificial intelligence” .

This development of AI applications by major technology drivers lead to the situation today where AI is relevant and ubiquitous in everyday applications.

1.4 Prominent AI Projects

Some AI projects have become milestones in AI history due to their public visibility. I shortly mention just three examples.

In 1997, *IBM Deep Blue* beat the chess world champion Garry Kasparov. This was a milestone since chess is considered one of the most complex board games.

See Fig. 1.3.

Fig. 1.3: IBM Deep Blue beats the chess champion Garry Kasparov

[1] http://bigdata-madesimple.com/12-famous-quotes-on-artificial-intelligence-by-google-founders

However on examining the computational algorithms used in the Deep Blue Application, people quickly realized the difference between the IBM approach and human intelligence. For people who believed in machines being potentially at least as intelligent as humans, this was a disillusionment (And since this belief still exists, this story is repeated for every successful AI project). For those practitioners whose goal is to build useful applications this was no disillusionment at all but an important milestone in the development of applications which exhibit behavior of human intelligence.

In 2011, IBM succeeded with another important AI project: *IBM Watson*. While Deep Blue targeted a most specific ability, namely playing chess, IBM Watson was able to answer natural language questions about general knowledge. The media highlight of this project was beating the human champions at the popular US quiz show Jeopardy! This was remarkable since not only questions about history and current events, the sciences, the arts, popular culture, literature, and languages needed to be answered but also play on words as well as execution speed and strategy needed to be considered.

See Fig. 1.4.

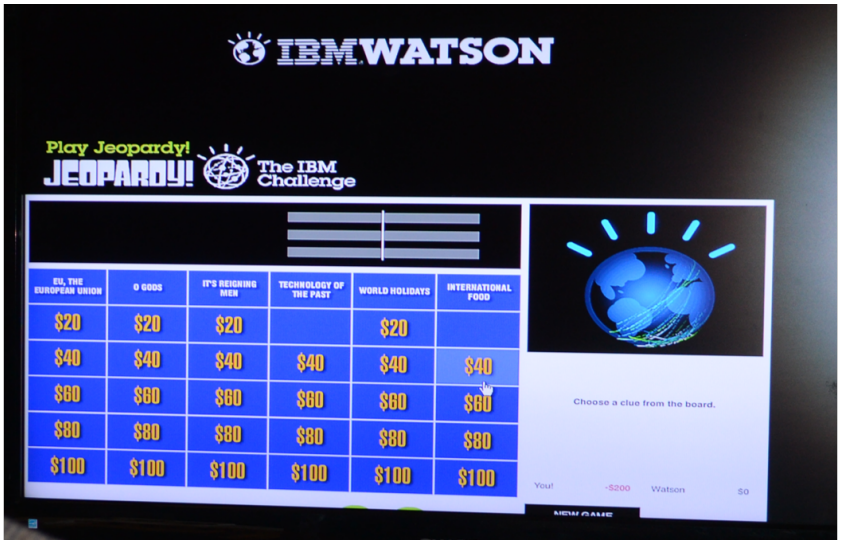

Fig. 1.4: IBM Watson for the Jeopardy! quiz show

Over the last years, Google has developed several prototypes of *self-driving cars* which have undergone intensive road testing. Google plans to make these cars available to the public in 2020 (as of 2015). The self-driving car is a major milestone in AI because, as an autonomous system, it not only has to process sensor data at real time but also has to plan and execute actions in the real world.

See Fig. 1.5.

Fig. 1.5: Google's self-driving car

To conclude, AI applications have made enormous progress in the last two decades - applications that were unthinkable in the late 20th century. All those applications exhibit behavior of human intelligence in certain situations (playing chess, answering questions, driving cars, etc.) without necessarily imitating human intelligence as such. Looking inside the implementation of such applications may be disappointing to people hoping to gain insights in very nature of human intelligence. They are IT applications and follow engineering practices of IT applications. And they may be useful in every-day's life.

The following chapters give insights into engineering such AI applications.

1.5 Further Reading

There are numerous publications on AI, scientific papers and text books. I present only a very short selection.

Stuart Russell's and Peter Norvig's book: "Artificial Intelligence: A Modern Approach" (Russell and Norvig, 2013) is the standard textbook on AI. It gives detailed insights in AI mechanisms and the theories behind them - issues which I do not focus on in this book. It may be used as an excellent source for further reading when you need to know more about those issues.

Marc Watson has written a number of books on practical aspects of AI: "Practical Artificial Intelligence Programming with Java" (Watson, 2013), "Practical Semantic Web and Linked Data Applications, Java, Scala, Clojure, and JRuby Edition" (Watson, 2010), "Practical Semantic Web and Linked Data Applications, Common Lisp Edition" (Watson, 2011). While they are some of the relatively few books dealing with the development aspects of AI applications, the focus is still somewhat different from the focus of this book. Watson introduces concrete frameworks an tools in concrete programming languages. In contrast, I rather give an overview of the tools available

in form of services maps and product maps, together with hints as to which tools to use in which context and how to integrate them. Insofar, also Watson's book may be used for further reading in case you have decided on specific languages and tools covered by him.

Also, there are a number of recommendable (free) online courses on AI - with a similar focus as Russell and Norvig's standard text book (2013). Examples are:

- Udacity: Intro to Artificial Intelligence[2]
- Sailor.org Academy: Artificial Intelligence[3]
- Sailor.org Academy: Advanced Artificial Intelligence[4]

1.6 Quick Check

Answer the following questions.

1. What does the term "Artificial Intelligence" mean?
2. What are main areas of AI?
3. Sketch the history of AI
4. In what sense is AI today relevant and ubiquitous?
5. Name a few prominent AI projects

[2] https://www.udacity.com/course/cs271
[3] http://www.saylor.org/courses/cs405/
[4] http://www.saylor.org/courses/cs408/

2. Knowledge Representation

AI applications are based on knowledge. Therefore, a central question to all AI applications is how to represent the knowledge relevant for the application domain.

Inspired by one of my own projects, the digital collection[1] of one of the leading arts museums in Germany, I will use arts as the sample application domain for this chapter and the remainder of the book.

See Fig. 2.1.

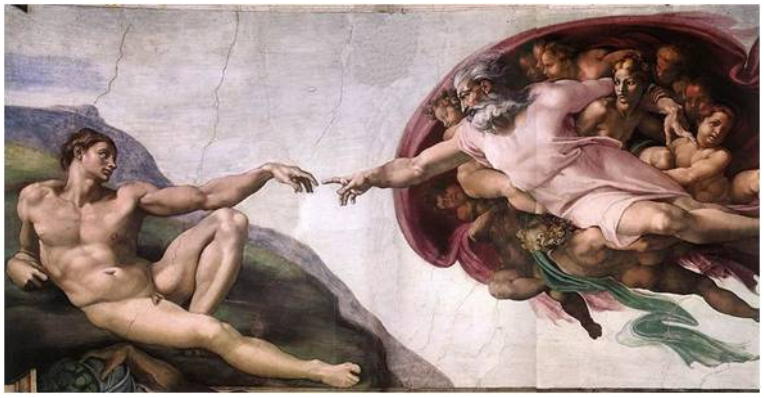

Fig. 2.1: Michelangelo (1475 – 1564): Creation of Adam, Sistine Chapel, Rome, Italy

Relevant questions for knowledge representation are:

- How to *represent knowledge*, e.g., "Michelangelo is a Renaissance artist"
- How to *reason over knowledge*, e.g., "Someone who has created paintings is an artist"
- How to *answer questions*, e.g., "Which Renaissance artists lived in Italy?"

2.1 Ontology

In this book, I use the term "ontology" for represented knowledge with the following definition.

An *ontology* is the representation of knowledge of a particular application domain, i.e., a *representation of the relevant concepts and their relationships*.

The term "ontology" was originally coined in philosophy and is now also used in Computer Science. See, e.g., (Busse et al., 2015). Let us look at some specific examples of knowledge about arts (Fig. 2.2).

[1] https://digitalesammlung.staedelmuseum.de/index.html

- Michelangelo was an Italian artist.
- He created many paintings, e.g., the „Creation of Adam“ in the Sistine Chapel, Rome.
- He also created sculptures like David.
- He belonged to the artistic movement of Renaissance.
- People who create paintings are painters.
- Painters are artists.

Fig. 2.2: Examples of arts facts

Those pieces of knowledge are represented informally as English sentences. Parts of those sentences are marked with different colors. The terms “artist”, “painting”, “sculpture”, and “artistic movement” are marked red (solid lines). They represent concept types a.k.a *classes*. The terms with dashed red lines are concept instances a.k.a. *individuals*: “Michelangelo” is an artist, “Creation of Adam” is a painting, “David” is a sculpture, and “Renaissance” is an artistic movement. Marked in blue are *relationships*: Michelangelo “created” David as well as Creation of Adam and he “belonged to” the High Renaissance movement. Finally, marked in green are general *rules*: Every individual who has created a painting is a painter; every individual belonging to the class painter also belongs to the class artist.

See Fig. 2.3.

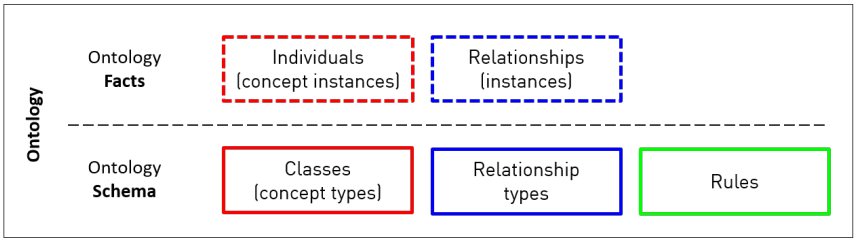

Fig. 2.3: Ontology facts and ontology schema

An ontology consists of *facts* and an ontology *schema*. Facts are based on the schema. Facts are individuals (concept instances, e.g., Michelangelo) and concrete relationships between those individuals (e.g., Michelangelo created David; dashed lines in the figure). The schema specifies the type level (solid lines in the figure). Classes are the concept types for individuals (e.g., artist for Michelangelo). Relationship types, e.g., “created” define the kinds of relationships that can be specified between individuals. Finally, *rules* are part of an ontology schema.

An ontology of some form is in the center of many AI applications. Extending the ontology, i.e., adding new concepts, means extending the AI application.

Ontology reasoning

An ontology is more powerful than a traditional relational database. In addition to the knowledge that is stated explicitly, knowledge may be derived via *reasoning*.

Consider the following explicitly stated facts and rules:

- Michelangelo is a person.
- Michelangelo has created the painting „Creation of Adam“.
- Every person that has created a painting is a painter.

From this explicitly stated knowledge, the following fact may be derived via reasoning:

- Michelangelo is a painter.

I leave it to the reader to find an explanation why the reasoning result is, in fact, correct. Take a while to think about it. What is the intuitive, common sense, explanation? Can you also formulate the explanation more formally? How would a computer program (a reasoning engine) have to be constructed in order to derive the new fact automatically?

Implementing a reasoning engine is beyond the scope of this book. This book focuses on engineering AI applications using off-the-shelf components like a reasoning engine (Much like a book on engineering JavaEE applications will not explain how the Java compiler is implemented). For further reading refer, e.g., to (Russell and Norvig, 2013).

Ontology Querying

Knowledge representation is no end in itself but a means to an end: answering relevant questions of the application domain. Therefore, ontology engines include a query interface that allow to formulate queries and retrieve results.

Consider the following questions and answers regarding the facts specified above:

- Q1: Who has painted “Creation of Adam”? A1: Michelangelo
- Q2: Has Michelangelo painted “Creation of Adam”? A2: Yes
- Q3: Is Michelangelo a painter? A3: Yes
- Q4: Which painters exist? A4: Michelangelo (and more painters, according to the facts specified)

The four questions are of different nature.

Q1 (Who has painted “Creation of Adam”?) can be answered by matching an explicitly stated fact (Michelangelo painted „Creation of Adam“) and returning the matching individual (“Michelangelo”).

Q2 (Has Michelangelo painted “Creation of Adam”?) is similar but expects a Boolean answer (here: Yes).

Q3 (Is Michelangelo a painter?) also expects a Boolean answer but cannot be answered by simply matching explicitly stated facts. Instead, ontology reasoning is required to derive the answer:

Michelangelo is a person and painted „Creation of Adam“; every person who has created a painting is a painter ⇒ Michelangelo is a painter. Hence, the answer is yes.

Q4 (Which painters exist?) also involves reasoning but expects a set of individuals as answer. In case the ontology only contained the facts mentioned above then the result would, indeed, be just {Michelangelo}. If, however, paintings by Raphael, Leonardo da Vinci, etc. were listed then the set would contain those artists as well.

Open versus Closed World Assumption

Q4 (Which painters exist?) leads to an interesting question: How to ensure that the ontology is complete, i.e., actually contains all painters?

In an ontology for our solar system it is relatively easy to list all planets (even though the definition of what is a planet may change over time: Pluto is, today, considered a dwarf planet only). In contrast, for the arts ontology it is next to impossible to list all painters that ever lived. Many of them are not known anymore. And who should decide who was a “real” painter and who just scribbled a few sketches?

Other critical questions are counting questions (How many painters exist?) and negated questions (Which painters have not created a sculpture?). The answers to those questions may be wrong if the ontology is incomplete – even if all facts in the ontology are, indeed, correct.

Therefore, the correct interpretation of ontology query results depends on assumptions that we make on the completeness of the ontology[2].

In the *Closed World Assumption (CWA)* it is assumed that the ontology is complete. This is a comfortable situation and makes the interpretation of query results easy. Assume e.g., that the arts ontology is restricted to one particular museum and is complete (closed world assumption). In this case, the answer to Q4 (Which painters exist?) may be “There are exactly the painters X,Y,Z for which we currently have paintings in our museum”; the counting query (How many painters are there?) may be answered “There are exactly n painters currently exhibiting in our museum”; the answer to the negated query (Which painters have not created a sculpture?) can be answered as “For painters X,Y,Z we have currently no sculpture in our exhibition”.

In the *Open World Assumption (OWA)* it is assumed that the ontology is not complete and new findings may be added as facts at any time. This does not mean that questions about enumerations, counting, and negations cannot be asked at all. However, the results must be interpreted differently than in the CWA. For example, the result to Q4 (Which painters exist?) can be interpreted as “painters that we know of …” instead of “all painters …”; The result of the counting question (How many painters are there?) can be interpreted as “there are at least n painters that we know of”; the answer to the negated question (Which painters have not created a sculpture?) may be interpreted as “for painters X,Y,Z there is currently no sculpture known”.

[2]The situation is, of course, further complicated if we even cannot assume that the specified facts are correct. There are various approaches for dealing with uncertainty (see (Russell and Norvig, 2013)). In practice, it is advisable to try to avoid this situation by quality assurance measures if at all possible.

The distinction between OWA and CWA is not technical - it is a matter of business application logic! It is about interpreting the results of an ontology query adequately. When developing an AI application it is essential to figure out under which assumption the ontology has been constructed in order to interpret results adequately.

2.2 Knowledge Representation Approaches

In the last sections, I introduced ontologies, ontology reasoning and querying informally. In order to use knowledge in an AI application one needs concrete formalisms and implementations of such formalisms. In this section, I briefly mention important knowledge representation formalisms. They have been developed over the years in AI research and they are, today, more or less used in practice. For more details see (Russell and Norvig, 2013).

Predicate Logic

Predicate logic is a mathematical formalism which is the foundation of many knowledge representation formalisms. Facts and rules are specified in form of predicates, quantifiers and Boolean operations. In the following example, `painted(p, x)` and `painter (p)` are predicates; The universal quantifier ("for all", denoted as an inverted letter "A") and the existential quantifier ("there exists", denoted as a rotated letter "E") are used; As a Boolean operation, the implication ("if … then", denoted as an arrow) is used.

See Fig. 2.4.

painted (Michelangelo, CreationOfAdam)
$\forall$p: ($\exists$ x: painted (p, x)) $\rightarrow$ painter (p)

Fig. 2.4: Predicate logic

Interpretation: Michelangelo painted "Creation of Adam"; Every person who painted something is a painter.

Frames

Frames is a knowledge representation mechanism which influenced early object-orientation and, hence, is familiar to most software engineers today.

See Fig. 2.5 for an example.

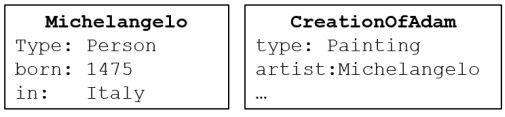

Fig. 2.5: Frame example

Interpretation: Michelangelo is a Person, born 1475 in Italy. “Creation of Adam” is a painting by Michelangelo. In frame systems, rules and reasoning can be implemented algorithmically in form of if-needed / if-added actions.

Semantic Nets

Semantic nets have been popular because of their intuitive graphical representation. In Fig. 2.6, individuals are represented as oval shapes, and relationships as arrows.

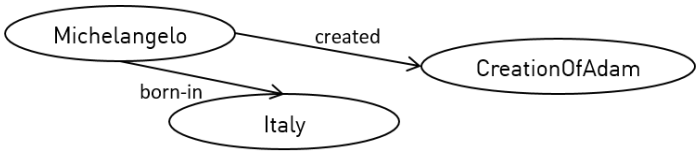

Fig. 2.6: Semantic net example

Interpretation: Michelangelo was born in Italy and created the painting “Creation of Adam”.

Rules

In rule-based languages, knowledge is expressed in form of facts and rules. In the following example (Fig. 2.7), `instance-of` and `painted` are predicates, `?p` and `?x` are variables, `person` and `painter` are classes, and `-->` denotes an implication. The two conditions on the left side of the implication are implicitly conjunctive, i.e., connected by the Boolean AND operator.

```
(?p instance-of person)
(?p painted ?x)
-->
(?p instance-of painter)
```

Fig. 2.7: Rules

Interpretation: If ?p is a person who painted something then ?p is a painter.

General-Purpose Data Stores

Rarely described in AI literature but commonly used in AI applications in practice are general-purpose data stores. Examples are relational databases and NoSQL databases including object databases, graph databases, key-value stores, search indexes and document stores.

I have co-edited a book on Corporate Semantic Web (Ege et al., 2015) in which 18 AI applications are presented which are in corporate use. More than half of the applications use general-purpose data stores for representing application knowledge. The architects of those applications made this decision due to performance reasons, ease of use, and for better integration into the corporate IT application landscape.

2.3 Semantic Web Standards

In the following sections I introduce RDF and SPARQL as sample languages and technologies for knowledge representation. Those languages have been standardized by the World Wide Web Consortium (W3C) as part of the Semantic Web initiative[3] and have gained some use in practice. I use those languages for examples in this book. However, neither the W3C standards nor other knowledge representation mechanisms and technologies can be considered as *the* de facto standard in today's AI applications.

Resource Description Framework (RDF)

RDF[4] stands for Resource Description Framework. RDF as well as the languages RDFS[5] and OWL[6] built on top of RDF are based on predicate logic and semantic nets.

I just explain RDF in a nutshell. For a good, practice-oriented introduction see (Allemang and Hendler, 2011).

Resources and URIs

An *RDF resource* represents anything which is relevant in an application domain, e.g., the individual "Michelangelo", the class "Painter", the relationship type "created", etc.

In RDF, every resource is identified by a *Uniform Resource Identifier* (URI[7]).

Example:

```
<http://dbpedia.org/resource/Michelangelo>
```

RDF Namespaces

Qualified names introduce namespaces to URIs in order to reduce typing effort, to increase readability, and to avoid name clashes.

Example of the URI above with qualified names:

```
@prefix dbpedia: <http://dbpedia.org/resource/> .
dbpedia:Michelangelo
```

[3] http://www.w3.org/2001/sw/
[4] http://www.w3.org/RDF/
[5] https://www.w3.org/2001/sw/wiki/RDFS
[6] https://www.w3.org/2001/sw/wiki/OWL
[7] http://www.w3.org/TR/webarch/#identification

RDF Triples

The main construct for specifying facts in RDF is a *triple* consisting of *subject, predicate*, and *object*. Subject and predicate must be RDF resources. The object may be an RDF resource but may also be a literal value in form of an XML data type[8].

Examples:

```
dbpedia:Michelangelo rdfs:label "Michelangelo"@en .
dbpedia:Michelangelo dbpedia-owl:birthDate "1475-03-06+02:00"^^xsd:date .
dbpedia:Michelangelo dbpedia-owl:movement dbpedia:Renaissance .
```

In the first example triple, the subject is `dbpedia:Michelangelo`, the predicate `rdfs:label` and the object `"Michelangelo"@en`. Every triple must terminated by a period.

Fig. 2.8 shows the example triples in displayed as a semantic net.

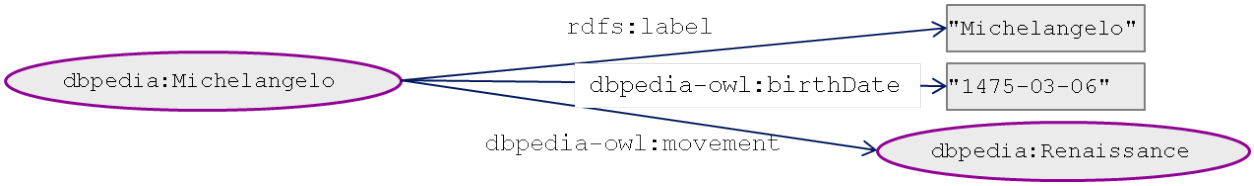

Fig. 2.8: RDF triples displayed as a semantic net

The triples may be read as: Michelangelo was born on 1475/03/06 and belongs to the artistic movement of Renaissance. In English applications, he is labeled "Michelangelo" (in other languages, other spellings may be preferred).

To avoid repeating the subject `dbpedia:Michelangelo` three times, an abbreviated notation may be used. The subject is stated only once and the triples with the same subject are combined via semicolons. A period terminates the group of triples.

The following example is semantically identical to the three triples above.

```
dbpedia:Michelangelo rdfs:label "Michelangelo"@en ;
  dbpedia-owl:birthDate "1475-03-06+02:00"^^xsd:date ;
  dbpedia-owl:movement dbpedia:Renaissance .
```

Classes

Classes represent concept types and are defined by RDF triples with `rdf:type` as predicate and `rdfs:Class` as object.

Example:

[8] http://www.w3.org/TR/webarch/#formats

```
dbpedia-owl:Person rdf:type rdfs:Class .
```

Meaning: Person is a class.

Individuals, i.e. concept instances, are specified by RDF triples with `rdf:type` as predicate and the class as object.

Example:

```
dbpedia:Michelangelo rdf:type dbpedia-owl:Person .
```

Meaning: Michelangelo is a person.

Properties

A relationship type is defined as an RDF property.

Example:

```
dbpedia-owl:birthDate rdf:type rdf:Property .
```

Meaning: `birthDate` is a property (relationship type).

An RDF property can be used as a predicate in an RDF triple.

```
dbpedia:Michelangelo dbpedia-owl:birthDate "1475-03-06+02:00"^^xsd:date .
```

Meaning: Michelangelo's birth date is 1475/03/06.

RDF and Object-Orientation

Classes and properties of RDF (and RDFS / OWL built on top of RDF) have some of similarities to classes and associations in object-oriented modeling and programming languages. However, there are also fundamental differences. For details see, e.g. (Allemang and Hendler, 2011).

Other Serialization Syntaxes

The RDF syntax introduced above is called Turtle[9]. It shall be noted that other RDF serialization syntaxes exist: N-Triples[10] and RDF/XML[11].

[9] http://www.w3.org/TR/2014/REC-turtle-20140225/

[10] http://www.w3.org/TR/2014/REC-n-triples-20140225/

[11] http://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/

Linked Data

Linked Data[12] is an initiative to create, interlink, and share ontologies for a wide range of application areas. Linked data is based on the W3C Semantic Web technologies introduced above. A large number of most comprehensive ontologies has been created and are publicly available – see the Linked Open Data Cloud in Fig. 2.9.

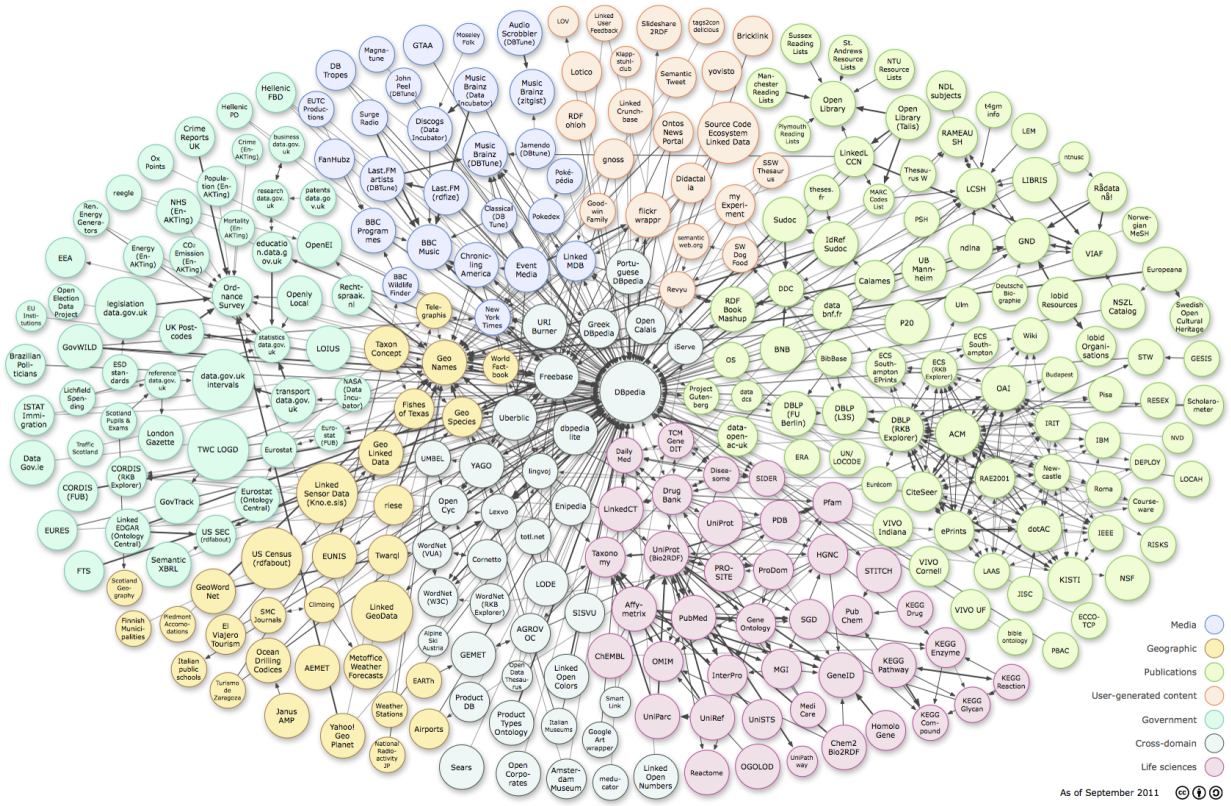

Fig. 2.9: Linked Open Data (LOD) Cloud

Prominent linked data ontologies are DBpedia[13], and YAGO[14]. Projects like WikiData[15], OpenStreetMap[16] and GeoNames[17] use different technology but follow the same idea.

Linked data is important: Due to the community editing and network effect, the coverage of topics is enormous. However, not every large linked data set is a suitable ontology for a concrete AI application. In section "Tips and Tricks" I give hints on ontology selection and creation.

[12] http://linkeddata.org/

[13] http://wiki.dbpedia.org/

[14] http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago//

[15] https://www.wikidata.org/wiki/Wikidata:Main_Page

[16] https://www.openstreetmap.org/

[17] http://www.geonames.org/

Example: Arts Ontology

In this book, I use an Arts Ontology in RDF as example. The Arts Ontology is a custom subset of DBpedia. DBpedia itself is a crowd-sourced community effort to extract structured information from Wikipedia and make this information freely available as an ontology. The Arts Ontology consists of ca. 1,000 paintings, 200 sculptures, 400 artists, 200 museums, 80 artistic movements, and 800 locations. The whole ontology contains about 4 MB of RDF code and was extracted from an DBpedia endpoint[18]. The extraction scripts can be found in the appendix.

Ontology Architecture

A good ontology, like a good IT application, has an explicit architecture. See Fig. 2.10 for a UML[19] class diagram illustrating the Arts Ontology schema.

[18] http://dbpedia.org/sparql
[19] http://www.uml.org/

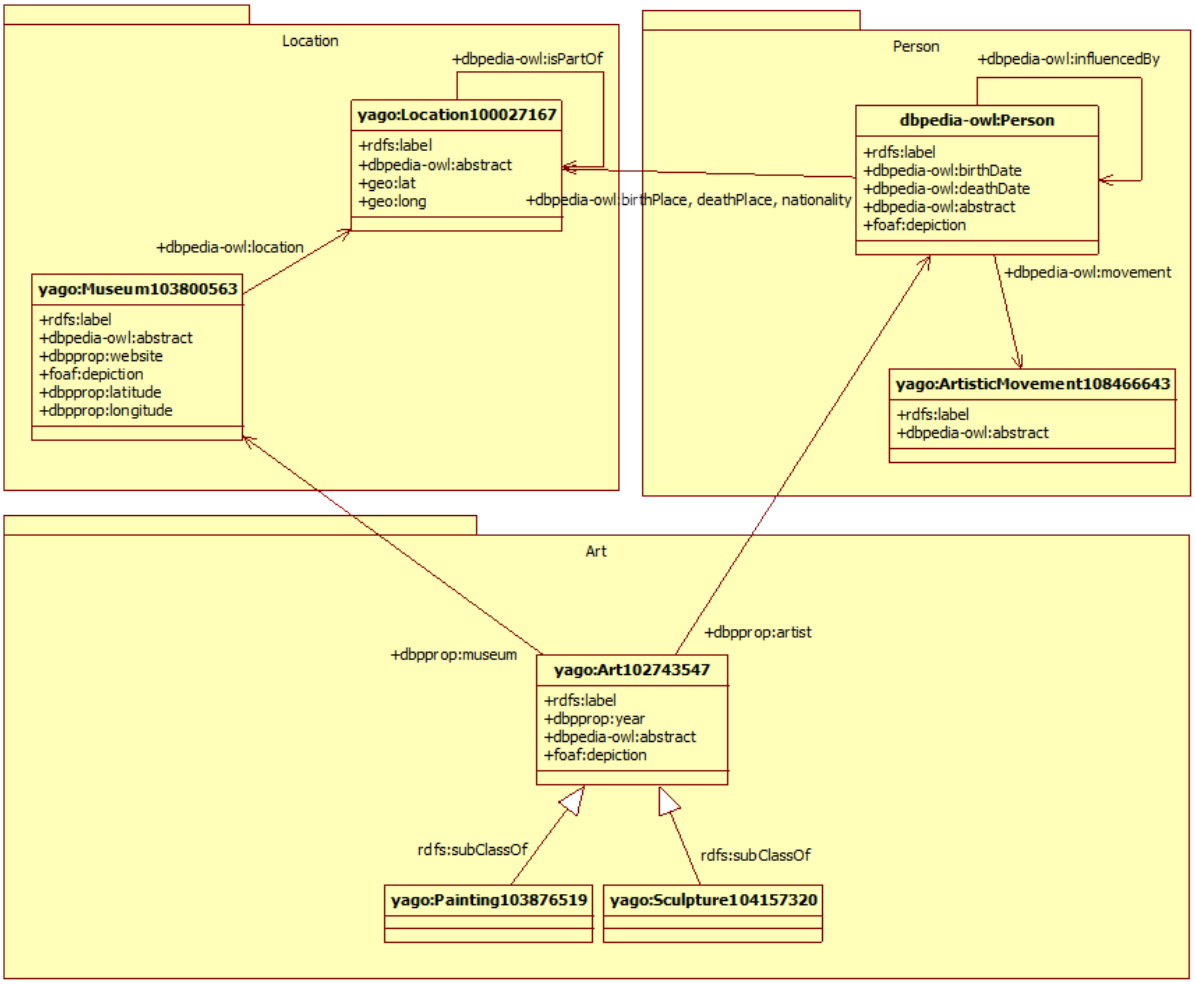

Fig. 2.10: Arts ontology schema

The Arts Ontology consists of three components: `Art`, `Person`, and `Location` (depicted as UML packages). The `Art` component contains artworks, in particular paintings and sculptures. The `Person` component contains artist and their artistic movements. The `Location` component contains museums and their locations.

The ontology classes are depicted as UML classes, e.g., `dbpedia-owl:Person` or `yago:Art102743547`. The namespaces and class names are extracted from DBpedia and YAGO and reflect the naming conventions of those ontologies.

RDF properties with literal values as objects are depicted as attributes of the UML classes, e.g., `rdfs:label`, `dbpedia-owl:birthDate`, and `foaf:depiction`.

RDF properties that link individuals are depicted as UML associations, e.g., `dbpprop:artist`, `dbpedia-owl:birthPlace`, and `dbpedia-owl:location`.

Ontology Editor

Protégé[20] is an open source ontology editor. Fig. 2.11 shows as an example, the entry of `dbpedia:Michelangelo` of the Arts Ontology in Protégé.

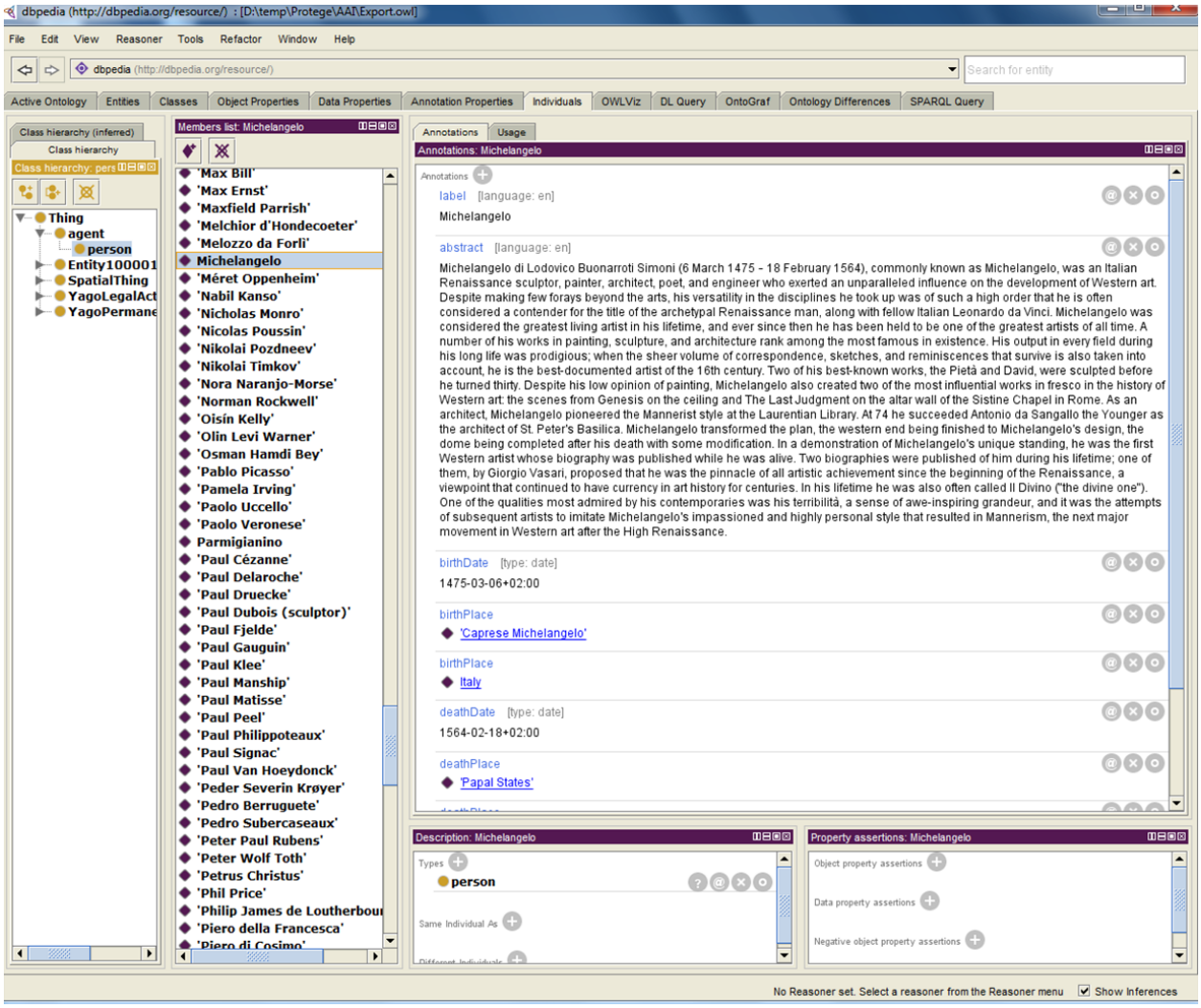

Fig. 2.11: Protégé

SPARQL

SPARQL[21] is the query language for RDF and is also standardized by the W3C.

[20] http://protege.stanford.edu/

[21] http://www.w3.org/TR/sparql11-query/

SPARQL Namespaces

SPARQL also supports namespaces but with slightly different a syntax than RDF. See, e.g., some relevant namespaces. The keyword `PREFIX` is used to define a namespace prefix which is separated by a colon from the full namespace.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
PREFIX dbpprop: <http://dbpedia.org/property/>
PREFIX yago: <http://dbpedia.org/class/yago/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
```

Simple SPARQL Queries

SPARQL uses the keywords `SELECT` and `WHERE` similar to SQL[22]. The query conditions are expressed as RDF triples in which variables can be used. Variables start with a question mark.

See e.g. the following SPARQL query resembling the question "What is Raphael's nationality?"

```
SELECT ?n
WHERE {
   dbpedia:Raphael dbpedia-owl:nationality ?n .
}
```

In this query, the variable for holding the nationality is `?n`. The query result is the set of all objects `?n` for which RDF triples `dbpedia:Raphael dbpedia-owl:nationality ?n` exist in the ontology. The result is one such object, namely the resource `dbpedia:Italy`.

Fig. 2.12 shows the query and its result in Protégé after loading the Arts Ontology.

[22] http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=53681

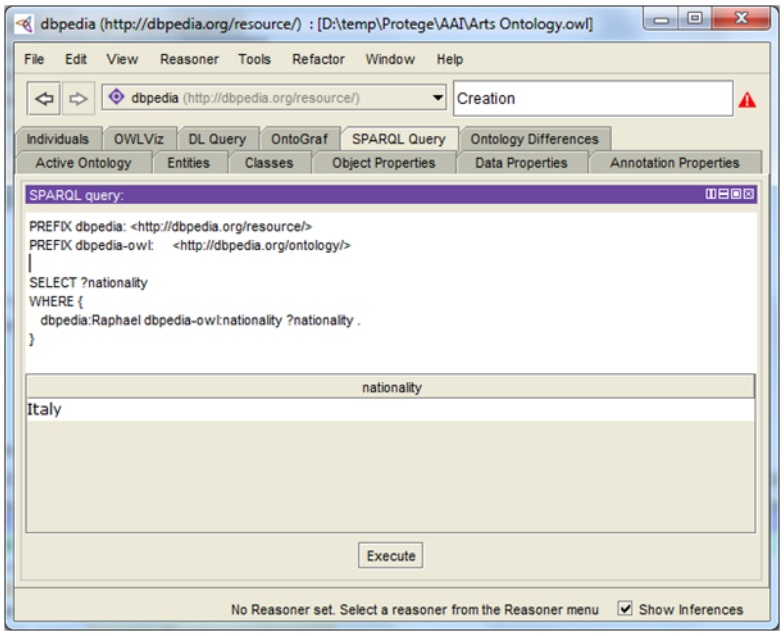

Fig. 2.12: A simple SPARQL query in Protégé

Since the Arts Ontology is a subset of DBpedia, you can execute the query in at the DBpedia SPARQL endpoint[23] (See Fig. 2.13). Try it yourself!

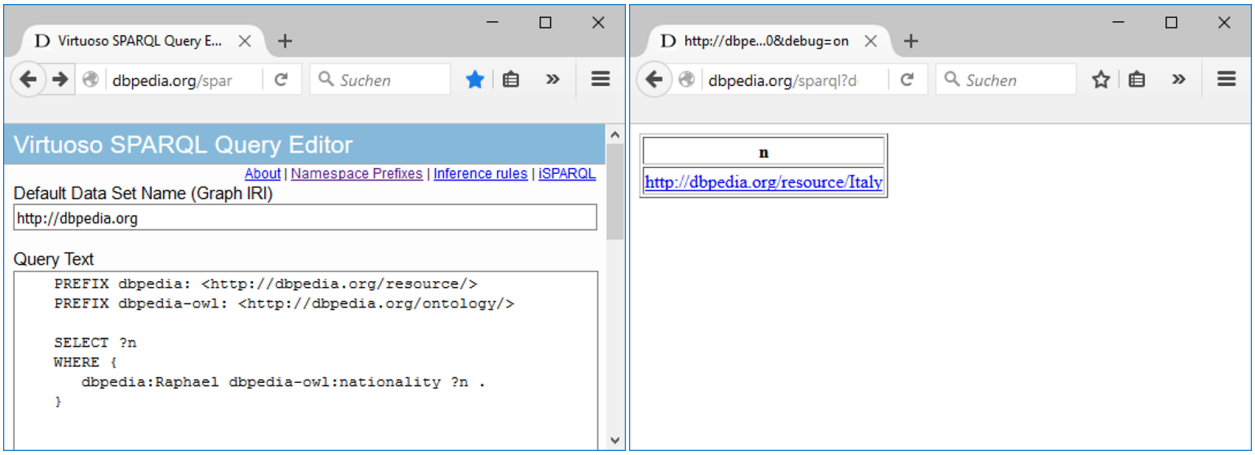

Fig. 2.13: A simple SPARQL query at the DBpedia SPARQL endpoint

Assume we are interested in Italian artists. This question contains two conditions: being Italian and being an artist. The respective SPARQL query, hence, contains two triples as a condition.

[23] http://dbpedia.org/sparql

```
SELECT ?p
WHERE {
   ?p rdf:type dbpedia-owl:Person ;
      dbpedia-owl:nationality dbpedia:Italy .
}
```

Implicitly, all triples in the `WHERE` clause of a SPARQL query are conjunctive, i.e., `AND` connected. The result of this query is the set of resources with this condition. It includes, e.g., `dbpedia:Michelangelo` and `dbpedia:Raphael`.

As you can see in this example, the abbreviated RDF Notation linking several triples with the same subject via a semicolon can be used in SPARQL, too.

Multiple Query Result Variables

When you are interested in multi-value query results, then multiple result variables may be specified in the `SELECT` clause of a SPARQL query.

The following query, e.g., lists artist with their corresponding nationality.

```
SELECT ?p ?n
WHERE {
   ?p rdf:type dbpedia-owl:Person ;
      dbpedia-owl:nationality ?n  .
}
```

The query result is a set of pairs `?p`, `?n`. See Fig. 2.14.

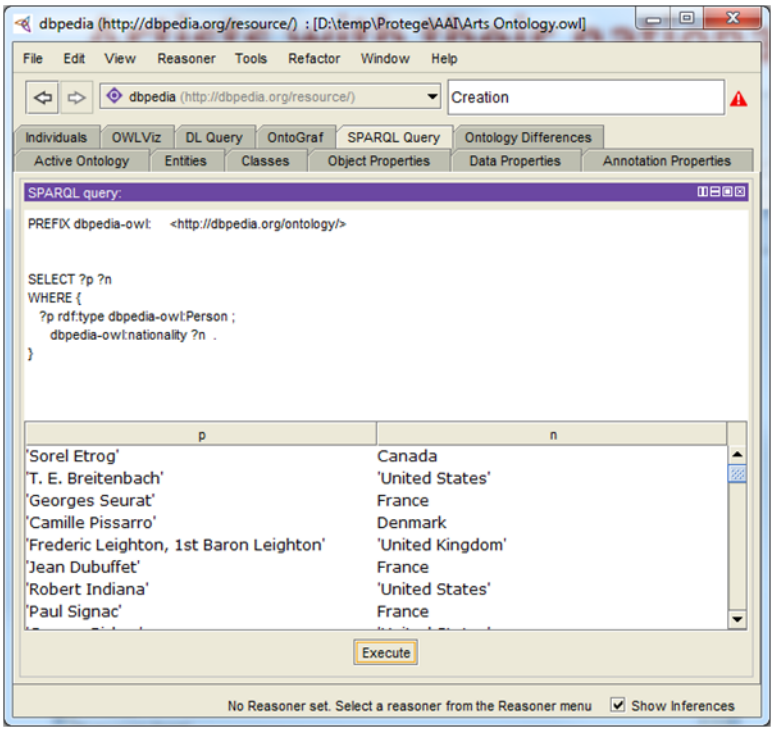

Fig. 2.14: Multi-variable query in Protégé

If all used variables shall be returned then `SELECT *` may be used as in SQL.

Distinct Query Results

As in SQL, `SELECT DISTINCT` avoids duplicates in the result set.

The following query lists the nationalities of all artists in the Arts Ontology.

```
SELECT DISTINCT ?n
WHERE {
   ?p rdf:type dbpedia-owl:Person ;
      dbpedia-owl:nationality ?n   .
}
```

You may try executing the query yourself.

Advanced Features

There are numerous advanced SPARQL features which are important for practical use but which I will not go into details in this book:

- Path expressions: an abbreviated notation for predicate / object chains

- Transitive closures (*): for querying arbitrary-length chains of the same predicate
- `ASK` queries: for checking a condition (Boolean result)
- `CONSTRUCT` queries: for generating new RDF statements (like production rules)
- `FILTER`: for expressing additional conditions, e.g., on datatypes
- `OPTIONAL`: for specifying optional values
- `EXISTS` / `NOT EXISTS`: for defining negation queries
- `GROUP BY`, `HAVING`: for aggregations
- `ORDER BY`: for sorting query results
- Subqueries: for nesting queries

For those and other features refer to the SPARQL specification[24].

2.4 Services Maps and Product Maps

Services maps and a product maps are diagrams that I present for each AI area in the respective chapter of this book. What are those maps and what can they be used for?

A *services map* depicts groups of functionality that products offer. For knowledge representation you may, e.g., need a knowledge editor at development time and a reasoning engine at runtime. So, "knowledge editor" and "reasoning engine" are services for knowledge representation.

Different tools and tool suites (commercial and open source) offer different services that often overlap. A *product map* maps the services to specific products. For example, Protégé is a knowledge editor. In fact, the product map depicts a table with tools as rows and services as columns. See Fig. 2.15 for an example.

	Integrated Environment	Knowledge Editor	API	Query Engine	Reasoner	Integration / Enrichment	Knowledge Base	Knowledge Resources
Protégé		*						
Virtuoso			*	*	*		*	
Sesame			*	*	*		*	
Apache Jena			*	*	*		*	
Tropbraid Suite	*	*	*	*	*	*	*	
DBpedia								*
YAGO								*

Fig. 2.15: Product table

[24] http://www.w3.org/TR/sparql11-query/

For each product map, such a table can be found in the appendix of this book.

Use of Services Maps and Product Maps

You may use services maps and product maps for selecting suitable products for an AI application development project. I recommend the following steps.

1. For a concrete project, mark the *services* which are *relevant*. For example, in a project where a new ontology is to be created, a knowledge editor is needed whereas if the ontology already exists, a knowledge editor is not required.
2. Consult the product map and retrieve products that cover the relevant services. Those are potential *product candidates*.
3. Now *select a product or a set of products*. This step, of course, requires a lot of expertise. In many organizations there are best practices for product evaluation processes. Usually, evaluation criteria are first defined and then the product candidates are evaluated against those criteria. If there are too many candidates, then a shortlist is prepared first.
4. If more than one product was chosen (best-of-breed approach) integrate the products for your solution.

Note: The integration of different products can be expensive. If there is a tool suite which matches all requirements, favor it over a best-of-breed solution. On the other hand, try to avoid vendor lock-in. Use products with open, standardized interfaces which can be replaced later if needed.

Knowledge Representation Services Map

Fig. 2.16 shows the knowledge representation services map.

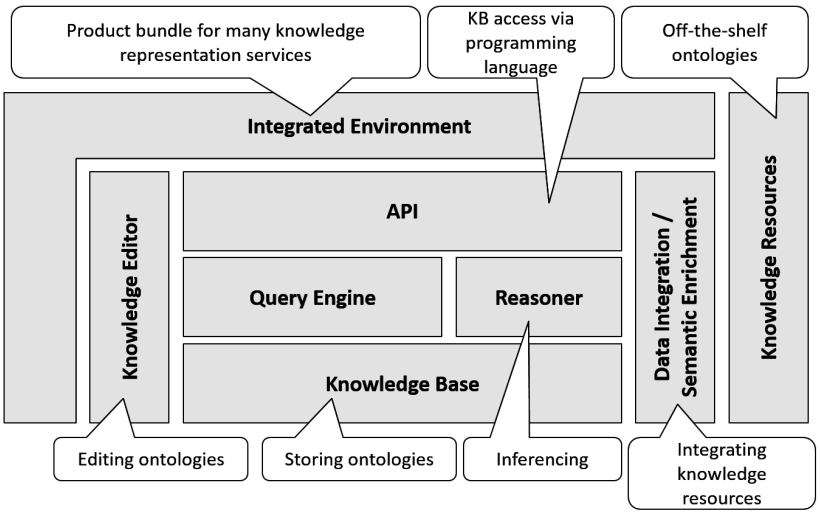

Fig. 2.16: Knowledge representation services map

- A *knowledge base* allows the storage and retrieval of ontologies, i.e. knowledge structures of all kinds. It is usually the core of an AI application.
- *Query engine* and *reasoning engine (a.k.a. reasoner)* usually come with a knowledge base product. But since they can often be plugged in and replaced, I have added them as separate services.
- Since AI applications are often implemented in a general-purpose programming language like Java, an Application Programmers' Interface (*API*) is needed to access to the knowledge base and its reasoners.
- A *Knowledge Editor* allows editing of ontologies. Ontologies may be imported / exported between knowledge editor (development time) and knowledge base (run-time). Standard formats like RDF may be used for importing / exporting.
- *Knowledge resources* include off-the-shelf ontologies like, e.g., DBpedia, that may be used in an AI application.
- *Data Integration / Semantic Enrichment* are services that allow integrating various knowledge resources or subsets thereof. For example, the Arts Ontology described above is a custom subset of DBpedia.
- *Integrated environments* are tool suites that bundle various development-time and run-time knowledge representation services.

Knowledge Representation Product Map

Fig. 2.17 shows the knowledge representation product map.

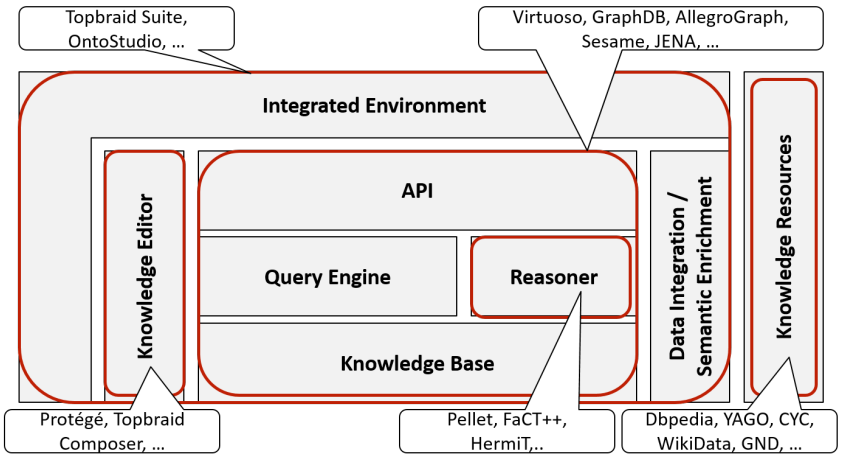

Fig. 2.17: Knowledge representation product map

Virtuoso, Owlim and JENA are bundles that include knowledge editor, reaonsing / query engines and Java APIs. Pellet, FaCT++, and HermiT are reasoning engines that may be plugged into other products. Protégé and Topbraid Composer are knowledge editors; Topbraid Suite and OntoStudio are integrated environments that include knowledge editors and runtime-components. Examples for knowledge resources are DBpedia, YAGO, WikiData, etc.

More products and details can be found in the appendix.

2.5 Tips and Tricks

Ontology - Make or Buy?

Make or buy? This is common a question in the technology selection phase of an IT project. It also often applies to the ontology in an AI application project.

As outlined in the section on Linked Data, there are hundreds and thousands of ontologies available, many with a public license, with thousands or even hundreds of thousands of facts each. An area with particularly well-established public ontologies are life sciences; See, e.g., The OBO Foundry[25] with ontologies for symptoms, diseases, medications, clinical procedures, genes / proteins, etc. etc. There are even *ontology search engines*, e.g., Swoogle[26] or Watson[27].

Surprisingly however, when being confronted with concrete requirements for an AI application, it turns out that rarely an off-the-shelf ontology is sufficient. In none of my previous projects in various areas (tourism, libraries, arts, software engineering and medicine) I could simply take and use an off-the-shelf ontology without adaptation. Other practitioners have made similar experiences (Ege et al., 2015).

I think Bowker and Star (1999) give a good explanation for this dilemma. They write: "Classifications that appear natural, eloquent, and homogeneous within a given human context appear forced and heterogeneous outside of that context". Different use cases require different structuring of the same knowledge.

Also, the quality of some public ontologies is mediocre. Consider, e.g., DBpedia and the following SPARQL query for finding all Renaissance artists.

```
1 select ?p where {?p rdf:type yago:RenaissanceArtists}
```

This query executed at the DBpedia SPARQL endpoint[28] (on 2016-01-10) results in only 16 (!) resources. `dbpedia:Michelangelo` is among them but, e.g, `dbpedia:Raphael` is missing. Instead, senseless results like `dbpedia:Leonardo_da_Vinci's_personal_life` are included.

Interestingly enough, the query

```
1 select ?p where {?p rdf:type yago:RenaissancePainters}
```

[25] http://www.obofoundry.org/
[26] http://swoogle.umbc.edu/
[27] http://watson.kmi.open.ac.uk/WatsonWUI/
[28] http://dbpedia.org/sparql

results in 550 resources. Aren't painters artists?!

When implementing the scripts for extracting the Arts Ontology (see above) from DBpedia, I put quite some effort in data cleansing measures, e.g., eliminating birth and death dates that are not valid, birth and death places that are no locations, artist's movements that are no artistic movements etc.

DBpedia is still a suitable source for the Arts Ontology example in this book. However, in a project for one of the leading German Arts museums (digital collection[29]), the use of DBpedia was out of question due to its quality deficiencies.

Of course, it is generally advisable to favor an off-the-shelf ontology over an custom-made one. Advantages are:

- Less costs of creating and maintaining the ontology
- Potentially higher quality due to contribution and use by many communities (as mentioned above, this is not always the case)

However, developing a custom ontology for a specific use case, e.g., within a corporation is not as expensive as one might think. Experience from practice shows that an experienced team can model about 1,000 concepts within 5 man-days (Hoppe, 2015).

Pre-Processing Off-the-Shelf Ontologies

When analyzing off-the-shelf ontologies for a concrete application use case, the ontologies' scope, structure and quality should be taken into account. Sometimes one or several ontologies can be identified that are quite suitable but do not fit perfectly. In this case a pre-processing step, much like the ETL (Extraction, Transformation, Loading) step in Data Warehouses, is recommendable. Ontology pre-processing may include the following activities:

- Transforming technical data formats, e.g., from tabular format CSV to RDF
- Transforming ontology schemas, e.g., from `yago:Art102743547` to `:Artwork`
- Quality enhancement, e.g., omitting birth dates that are not valid
- Integrating multiple ontologies, e.g., removing duplicates

This pre-processing step may be supported by services of type "Data Integration / Semantic Enrichment" in the Services Map. From my point of view, this important step is not discussed enough in the AI literature. See also the chapter on AI Application Architecture.

One final remark regarding selecting off-the-shelf ontologies: do not worry too much about technical formats. Converters between XML, CSV, XLS, RDF, database dumps, etc. are available open source or can easily be developed.

[29] https://digitalesammlung.staedelmuseum.de/index.html

Deciding on Technology

There are numerous mature products for knowledge representation available – commercial and open source. So, when it comes to selecting knowledge representation technologies and products, there is no make versus buy decision to be taken – just like you do not implement your own database management system for a business information system.

In the section on Services Maps and Product Maps I recommend a general method for selecting suitable product.

Here I give some tips and tricks regarding selecting knowledge representation products:

Check carefully what is *really* needed in your application use case. In many practical use case scenarios, much less is required than what traditional AI frameworks offer. In AI literature, there is much discussion about the expressive power of the OWL full standard compared to OWL light and other standards. In most of my projects, none of the OWL reasoning facilities were required at all. For many application use case scenarios, reasoning over hierarchies is enough (e.g., an artist who lived in Florence also lived in Italy since Florence is in Italy).

On the other hand, *high performance* requirements are common in real-world applications. However, traditional AI knowledge representation products exhibit poor performance due to their sophisticated reasoning facilities. Therefore, high-performance solutions like RDBMS and NOSQL data stores including search index solutions are recommended. Requirements like, e.g., reasoning over hierarchies can often easily be implemented using clever indexing strategies. Those solutions also deliver additional services like, e.g., full-text search which are not provided by classic AI knowledge representation products.

So, my recommendation is to not unnecessarily narrow the product candidates to traditional AI and Semantic Web technology when implementing AI applications.

2.6 Quick Check

Answer the following questions.

1. What is the purpose of knowledge representation?
2. Give examples for classes, individuals, relationships, and rules.
3. What is an ontology?
4. Name different knowledge representation approaches.
5. What does reasoning mean? Give an example. How does it work?
6. Explain the open word assumption and closed world assumption. What are the differences? What are implications of the assumption made?

7. What is a resource in RDF? How are namespaces used? What is an RDF triple?
8. How are classes declared in RDF? How are individuals (instances) assigned to classes?
9. How are properties declared in RDF? How are properties used?
10. Which serialization syntaxes exist for RDF?
11. What is linked data? Give examples of off-the-shelf ontologies.
12. How are simple queries structured in SPARQL?
13. How is the result set of SPARQL queries structured?
14. What advanced features does SPARQL offer?
15. What are services maps and product maps? How can they be used?
16. Name knowledge representation services.
17. Name a few products that implement those services.
18. How to select ontologies for a concrete application use cases?
19. How to integrate various off-the-shelf ontologies?
20. How to select knowledge representation products for a concrete AI application project?

About the Author

Bernhard G. Humm is a professor for software engineering and project management at the Computer Science Department of Hochschule Darmstadt - University of Applied Sciences, Germany. He received a Ph.D. from the University of Wollongong, Australia and the degree Dipl.-Inform. from Kaiserslautern University, Germany. His research focus is on semantic applications, software architecture, and programming languages. He is a member of the board of directors of the institute of applied informatics, Darmstadt (aIDa) and Ph.D. coordinator. He is running several national and international research projects in co-operation with industry and research organizations and is publishing his results regularly. Before re-entering university as a professor, he worked in the IT industry for 11 years, as a software architect, chief consultant, IT manager and head of the research department of a large German software company. His clients were in the industry sectors finance, tourism, trade, and aviation.

Contact:
Prof. Bernhard G. Humm, Dipl.-Inform., Ph.D. Computer Science (AUS)
Hochschule Darmstadt - University of Applied Sciences, Computer Science Department
Haardring 100, 64295 Darmstadt, Germany
bernhard.humm@h-da.de, www.fbi.h-da.de/∼b.humm