

Constructing an Explanation Ontology for the Communication and
Combination of Partial Explanations in a Federated Knowledge
Environment

Cornelis Bouter
Student number: 4030877

Daily supervisor: Barry Nouwt, MSc
First supervisor: prof. dr. mr. Henry Prakken
Second examiner: prof. dr. Rosalie Iemhoff

May 29, 2019

Contents

1	Introduction	1
1.1	Problem	2
1.2	Purpose and goals	3
1.3	Research questions	7
1.4	Structure	9
2	The domain: knowledge base explanation	10
2.1	Machine learning	11
2.1.1	Taxonomies	11
2.1.2	Extracting concepts	18
2.2	Explanation in other knowledge bases	20
2.2.1	Rule-based expert systems	20
2.2.2	Bayesian networks	21
3	Insights from the social sciences and philosophy	26
3.1	Insights from the social sciences	26
3.2	Philosophical prescriptions	27
4	Towards a reference ontology	30
4.1	Purpose and intended use	30
4.2	Requirements	31
4.3	Ontology modularisation	31
4.4	Competency questions	32
4.4.1	Fact explanation	33
4.4.2	Explanation hierarchy	33
4.4.3	Knowledge bases	34
5	The technique: Description Logic	35
5.1	Syntax and semantics	35
5.2	Additional expressive power	36
6	Reference ontology design choices	39
6.1	Existing explanation ontologies	39
6.2	Fact explanation	41
6.2.1	Explanandum ontology	42
6.2.2	Explanans ontology	42

6.2.3	Tree-based structures	44
6.2.4	Rule-based structures	45
6.2.5	Feature importance, salience maps and prototype selection	46
6.3	Knowledge base ontology	47
6.4	Explanation tree ontology	48
6.4.1	Descendant relations	50
6.4.2	Revisiting rule-based explanation	50
7	Operational ontology and implementation	53
7.1	Serialisation	53
7.2	OWL profiles	55
7.3	Implemented ontology	56
7.3.1	Explanation ontology	57
7.3.2	Mortgage domain ontology	59
7.3.3	ExplanationTree individual	64
8	Ontology testing with competency questions	68
8.1	Verification and validation	68
8.2	SPARQL	69
8.3	Competency questions	70
8.3.1	Explanation hierarchy	70
8.3.2	Fact explanation	71
8.3.3	Knowledge bases	75
9	Proof of concept	78
9.1	Explainable Plasido	78
9.1.1	Generic	79
9.1.2	Domain specific	82
9.2	Flow of control	84
10	Conclusion	86
10.1	Discussion	88
10.2	Future research	88
	Bibliography	90
	Appendices	94
A	SPARQL queries	94

Acknowledgements

First of all, I would like to thank Barry for our weekly discussion sessions, resolving numerous Maven errors, and beating me at pingpong. A lot of thanks also goes out to everyone else at Connected Business, especially my fellow interns Willem and Merle, for making me enjoy my stay at TNO.

Secondly, I would like to thank Henry and Rosalie for the supervision and examination. Henry, you never failed to pinpoint the weaknesses in my writings. Rosalie, although the examination part is relatively small, you helped with some personal remarks along the way.

Finally, I would like to thank some people who contributed in other ways: Fons for *matsen*, my parents for support, and Kimberly for being *lief*.

This work was conducted using the Protégé resource, which is supported by grant GM10331601 from the National Institute of General Medical Sciences of the United States National Institutes of Health.

“We must be systematic, but we
should keep our systems open.”

Alfred North Whitehead,
Modes of Thought (1938)

Chapter 1

Introduction

This research was performed at the Data Science department of TNO as part of the Early Research Program (ERP) Applied Artificial Intelligence. The Data Science department consists of three groups: Explainable Data Science (EDS), Responsible Data Science (RDS) and Connected Business (CB). The EDS group develops methods to interpret data science algorithms, whereas RDS is concerned with ethical issues such as anonymising data. The CB group, where our research was performed, performs research on *interoperability*, i.e. on developing tools and methods to increase the ease and efficiency of linking and sharing data between various sources. As the title of our thesis suggests, our research is also closely related to developments in the EDS group. We place it at the intersection of explainable data science and interoperability.

Various human decisions are increasingly guided by automated processes. Especially sensitive decisions such as the release of a suspect on parole (Angwin et al. 2016), the diagnosis of a doctor, the decision to allow a potential house buyer a mortgage (Kvamme et al. 2018), and many more are based on the output of *knowledge bases* (KBs). The KB that guides the decision must be understood in its broadest sense. It can indicate a rule-based expert system, but it can also indicate any kind of machine learning model, a relational database, a Bayesian network, or another computational model. After all, the collection of entries in a database, the probabilities of a Bayesian network, and the decision boundary learned by a machine learning model can all be understood as a type of knowledge that has been extracted either manually by a domain expert or automatically by an algorithm from domain data. We will call the plural, an interconnected set of knowledge bases, a *federated knowledge environment*.

During this thesis we will employ a running example of a bank that uses several knowledge bases to decide whether to approve a mortgage application. The main KB is a machine learning model comparing the applicant to previous applicants. It outputs a prediction whether the person's mortgage application should be approved. Two other KBs are used to preprocess the data. First, a machine learning model infers the current state of the economy from real-time stock rates. Second, a Bayesian network predicts the job opportunities of the applicant given his or her job field, annual salary, and working experience. The federated knowledge environment is configured as shown in figure 1.1. The root KB deciding on a mortgage application uses the current state of the economy and the applicant's job opportunities as input, together with data on the applicant that did not need preprocessing.

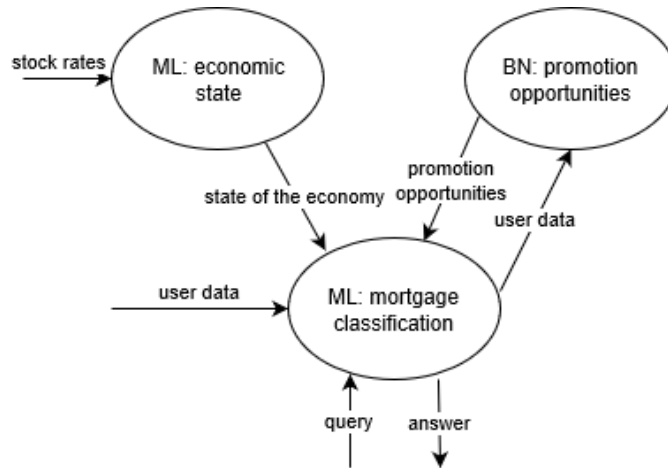


Figure 1.1: Configuration of the running example

1.1 Problem

The need to extract an explanation from the KB when it is applied on sensitive use cases has already become apparent in the explainable AI (XAI) and explainable machine learning literature (DARPA 2016; Lipton 2016). It can have far reaching effects on a person's life when an incorrect or unjust decision is made during a mortgage application procedure. However, the machine learning models that guide these decisions, are widely known as “black boxes”, because they are notoriously difficult to interpret. In the same way, the probabilities of a Bayesian network are hard to interpret for non-specialists (Timmer et al. 2017). On the other hand, KBs purely consisting of rules are usually judged to be interpretable, since a rule is a native concept to a non-specialist (Guidotti et al. 2018). However, a KB consisting of hundreds of rules quickly loses its interpretability (Lipton 2016). In all these cases, if the KB would reject the mortgage application, the developer nor the banking company would be able to interpret the classification beyond the output value.

Various explanation algorithms have in the last few years been developed because of the increasing abundance of machine learning applications (Biran and Cotton 2017). Our research will follow this trend, so we will mainly consider explainable machine learning. It will also include Bayesian networks (BN) for two reasons. First, we already mentioned that BNs require explanation algorithms, because they are difficult to interpret. Second, providing a solution that works for both machine learning and Bayesian network explanation gives evidence that the solution works for an arbitrary knowledge base. The goal of interoperability also is to connect various systems regardless of their type. We include expert systems for another two reasons. First, the TNO framework we use internally operates as a rule-based reasoner. Second, expert systems are among the oldest AI techniques, so various methods at explanation have already been developed (DARPA 2016).

An intuitive explanation algorithm is the *feature importance* algorithm. Given the mortgage prediction to reject, we might ask the model for the decisive feature. If we received an answer that either the applicant's job opportunities or monthly wage is the

most important feature, we would most likely consider it a correct prediction. On the other hand, if the person's surname or ZIP-code would turn out to be the deciding feature, we would have reason to manually review the application and to adjust the model. Given our domain knowledge, we know that a person's ZIP-code or surname is not a correct predictor for mortgage defaulting.

Without an explanation algorithm we would not have the possibility to interpret the classification beyond its face value. Although interesting, our research does not attempt to improve the explanation of a single algorithm. We will improve the explanatory power by integrating the existing explanations of various sources. We need to provide a solution where the whole explanation explains more than the sum of the partial explanations.

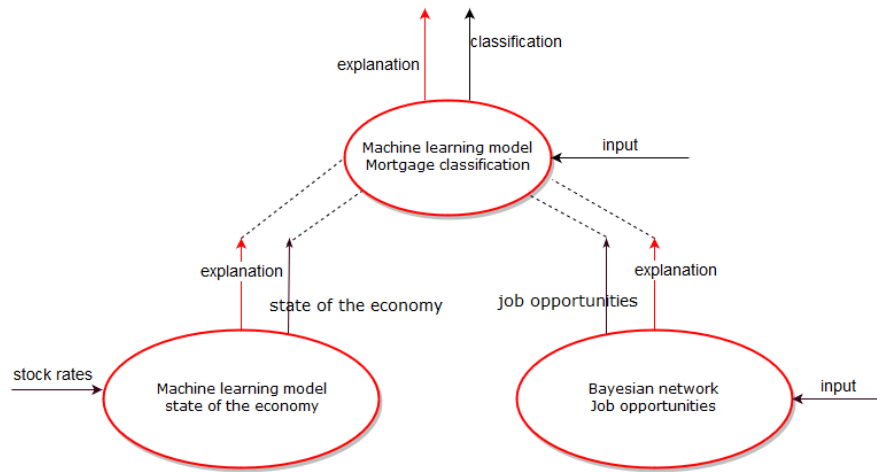
The explanation algorithms that have been developed thus far only consider the output of a *single* knowledge base. However, our running example is a federated knowledge environment consisting of three knowledge bases predicting the correct mortgage application decision, the state of the economy, and the applicant's job opportunities, respectively. The explanation of a prediction should include all three knowledge bases. Without any work on interoperability, our system would resemble figure 1.2a. Both the output and explanation of a KB have to be manually linked to the upper KB, resulting in an inefficient and fault-sensitive procedure. The Connected Business group at TNO already developed Plasido, a knowledge engine that allows for several independent KBs to be interconnected (Nouwt 2016). The current state of Plasido is given in figure 1.2b. The output of each KB is automatically used as an input for the appropriate KB one level higher.

The problem we face is that Plasido has no functionality to communicate an explanation. Any explanation a connected KB may produce is not automatically passed on to the upper KB. Therefore, the process of combining the explanation is sensitive to human error and lacks a well-defined standard. Moreover, the explanation of a single KB may be crucial to the explanation of the whole system. For example, the *feature importance* algorithm may return that the economic depression is the decisive feature. If we could not give an explanation *why* the economy is predicted to be in a depression, we still cannot explain the classification as a whole. If we cannot configure the knowledge bases as shown in figure 1.2c, the lack of interoperability leads to insufficient explanatory power.

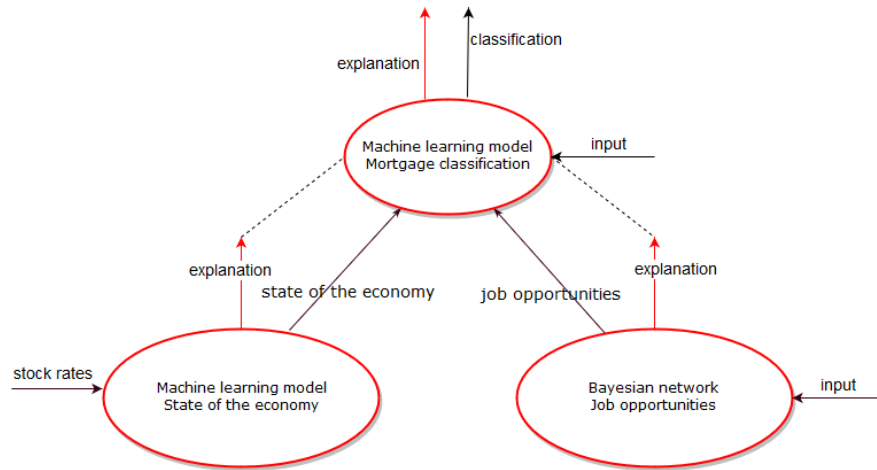
1.2 Purpose and goals

Our goal will be to extend Plasido with functionality to automatically communicate and combine explanations from the connected KBs. Semantic technology in the form of a rule base and a reasoner are employed to combine the various KBs into an integrated whole, such that a well-formed query automatically consults all connected KBs (Nouwt 2016). In effect, the knowledge residing in the individual KBs is combined to form a federated knowledge environment. Plasido is agnostic to the type of KB that is connected. Therefore, it is perfectly suited for the various types of KBs we already identified in section 1.1.

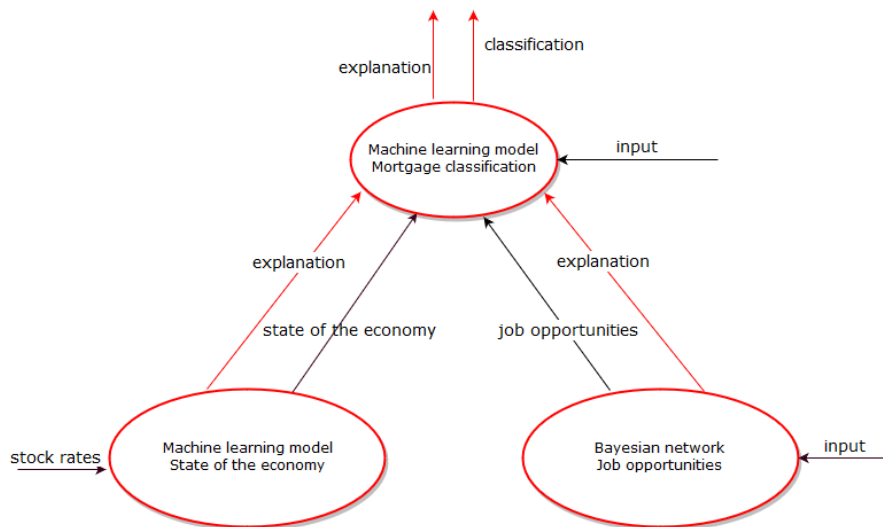
Figure 1.3a shows that, as of yet, Plasido has no functionality to communicate explanations, but can only provide an answer. Therefore, to employ Plasido in situations where an explanation is required, it should be expanded with functionality to commu-



(a) Before interoperability



(b) Current state



(c) Future state

Figure 1.2: Three states of explainability software systems. The dashed lines indicate that the operation has to be performed manually instead of automatically.

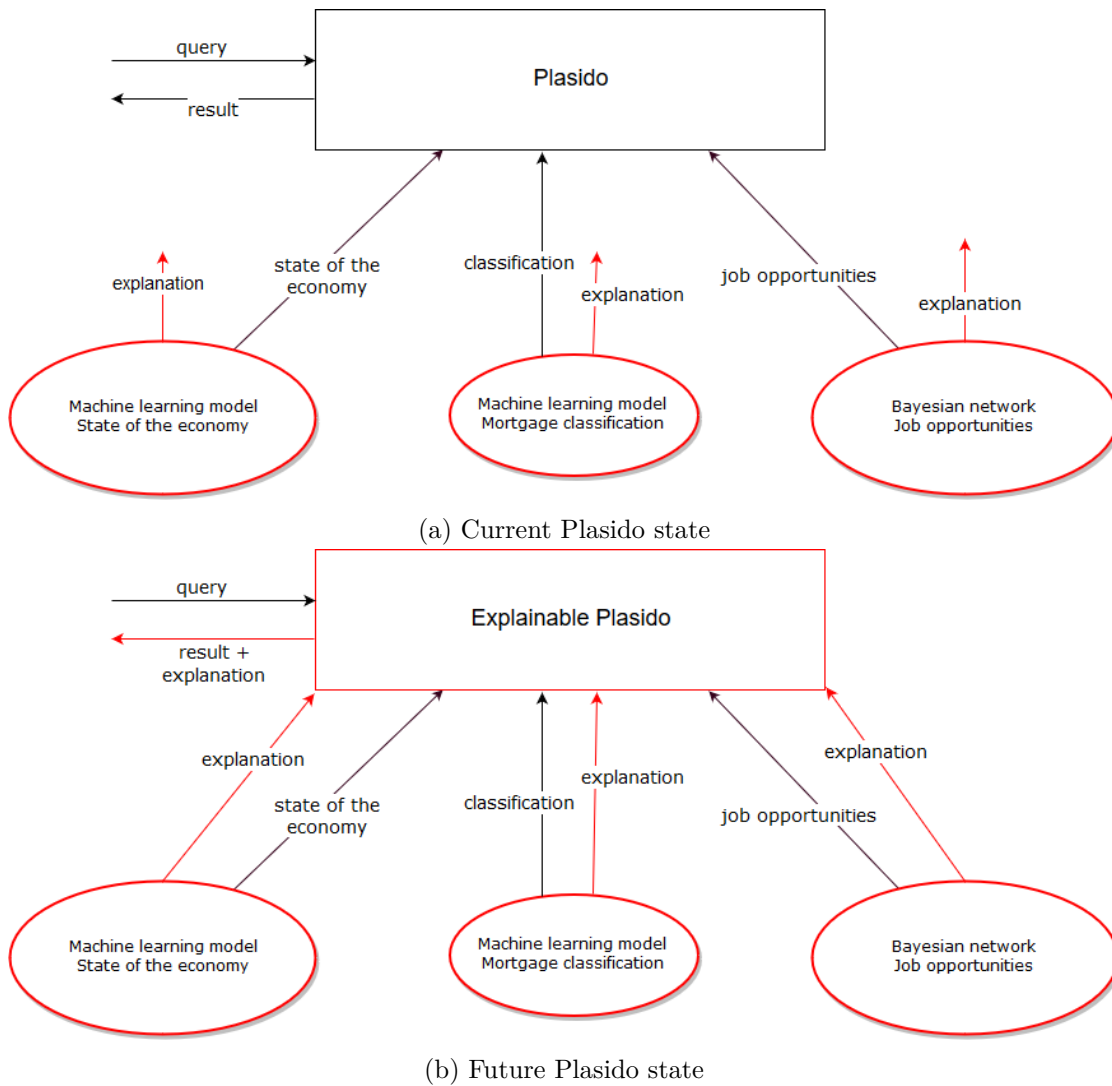


Figure 1.3: The current and future state of the Plasido engine

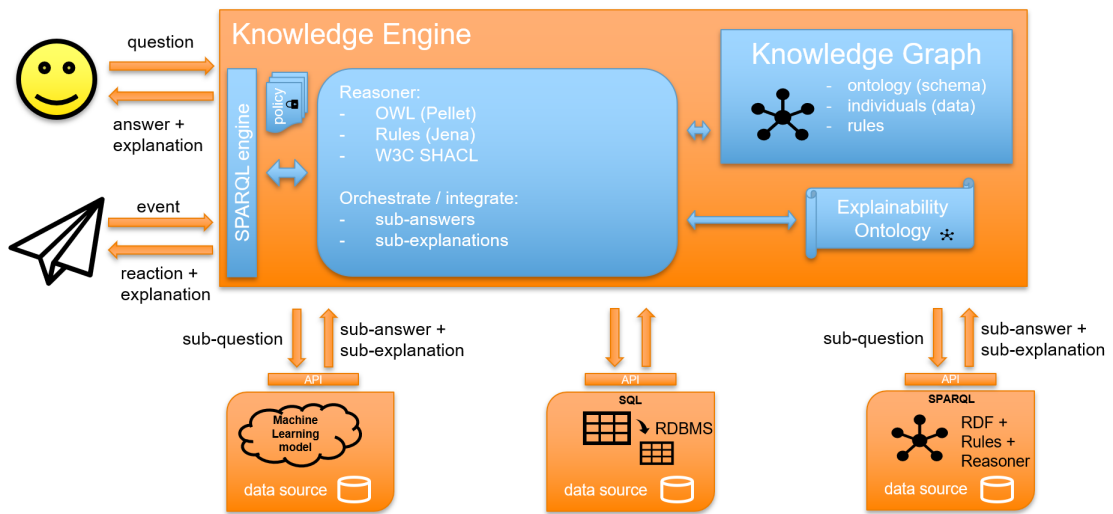


Figure 1.4: Plasido architecture

nicate and combine the explanation the external KBs may give. Figure 1.3b shows our goal state where each single explanation is automatically processed by Plasido.

A detailed schematic view of Plasido is given in figure 1.4. Our research will concern itself with the *explainability ontology*. An ontology is usually defined as an “implementation of a shared conceptualisation” (Gruber 1993). We can break this definition down into three parts:

Implementation: The ontology has to be written in a formal language, because the ontology has to be computer-readable. The Web Ontology Language¹ (OWL) is the W3C-recommendation for the implementation of ontologies, and the only standard that is widely applied. So, one of our deliverables will be a formalisation of the ontology in OWL.

Shared: An ontology defines a means of communication. It is useless if some KB would not adhere to the ontology and another KB would. That would be similar to humans each speaking a different language. Every KB in the application should understand and speak the language of the ontology.

Conceptualisation: The ontology is a systematic description of the domain, dividing it in concepts, relations, and individuals.

In short, the ontology will define the communication between Plasido and the individual KBs. If the ontology were not shared between Plasido and the individual KBs they would not understand each other. The conceptualisation defines what the content of the messages may be.

Several procedures for the construction of ontologies have been developed (Obrst et al. 2007; Vrandečić 2009). We will follow the guidelines for ontology construction of SAbiO: the Systematic Approach for Building Ontologies (Falbo 2014). This way

¹<https://www.w3.org/OWL/>

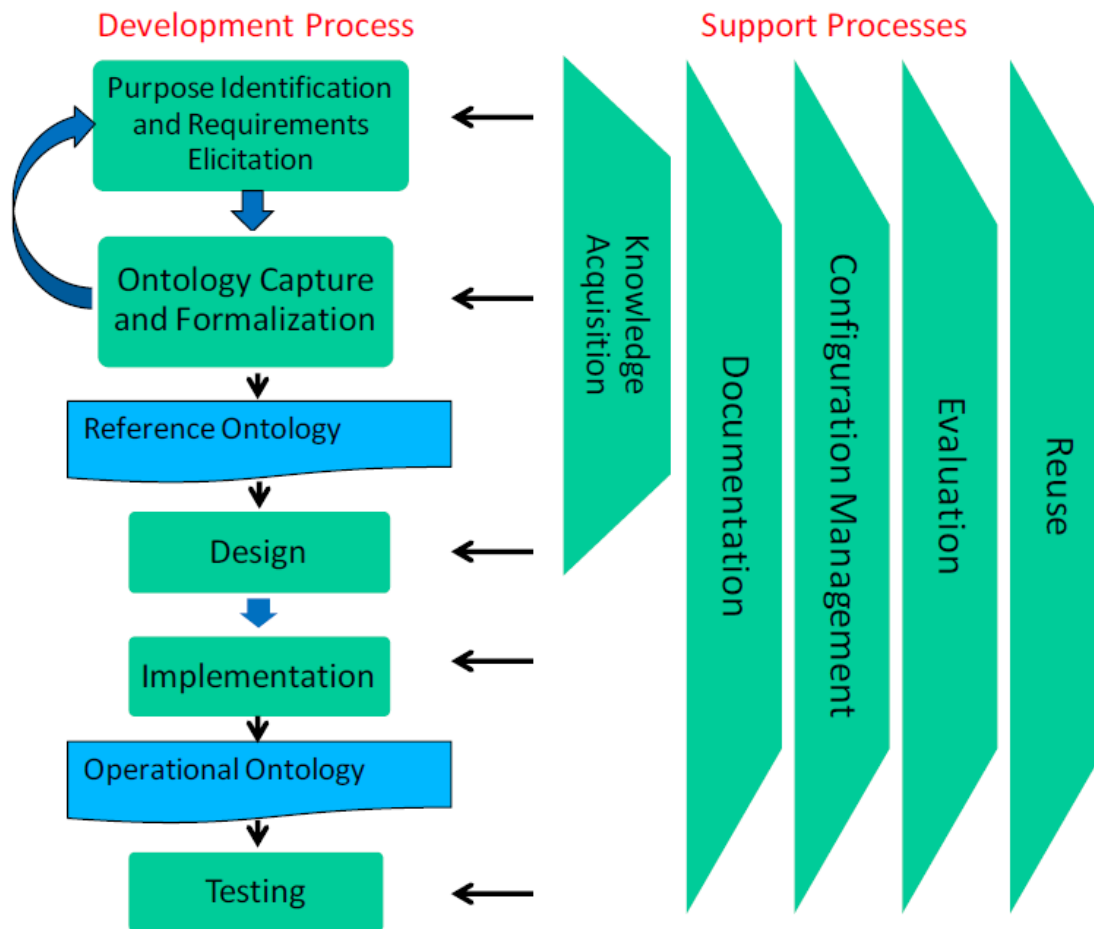


Figure 1.5: SABiO's processes (Falbo 2014)

we use previous researchers' experience at ontology construction. First of all, SABiO is intended for the construction of domain ontologies, rather than high-level foundational ontologies. Our ontology is also intended for a specific domain: knowledge base explanation. A major part of SABiO is eliciting domain knowledge from the literature, which is a task we also need to undertake. Second, SABiO grounds the domain ontology in existing (foundational) ontologies to increase reuse. Third, figure 1.5 shows that SABiO distinguishes between a *reference* ontology and an *operational* ontology. The former is a conceptualisation of the domain that is as precise as possible. The latter is an operationalisation where precision is traded for computational flexibility.

1.3 Research questions

Several research questions will guide the process. Our main research question is:

Main Question How do we leverage ontologies to integrate the various explanations of a federated knowledge environment?

The explanatory capacity of the federated knowledge environment needs to be more

than just the sum of the explanations of the individual knowledge bases. The main question can be divided into two separate processes: 1) the construction of an ontology for explanation and 2) the integration of the ontology into Plasido.

The processes of SABiO, especially the distinction between the reference ontology and the operational ontology, help us define the subquestions on the construction of the ontology. We begin with getting a grasp of the domain.

Subquestion 1 Which types of explanations are produced by knowledge bases?

Subquestion 2 Which types of explanations are useful according to philosophy and social theory?

The reference ontology functions as the ideal conceptualisation precisely describing the domain. It is constructed through a structured review of the relevant literature. The process of SABiO instructs us to answer four questions in preparation of constructing a reference ontology. The first question concerns the purpose and goals of the ontology, which we have already answered in this chapter. The remaining questions are given in subquestions 3.1 through 3.3. The competency questions of subquestion 3.3 are intended to specify the information that can be extracted from the ontology. They serve as both a proof of correctness and a tool for inspection.

Subquestion 3 What is the appropriate reference ontology?

Subquestion 3.1 What are the requirements we elicit from the literature?

Subquestion 3.2 How can we modularise the ontology?

Subquestion 3.3 Which competency questions characterise the information of the ontology?

Then, the reference ontology has to be implemented into a formal computer-readable language. This process usually involves a trade-off between exactly implementing the reference ontology and keeping desirable computational properties. When imposing too much restrictions the ontology becomes undecidable. However, we want to use the ontology to let a reasoner infer facts previously not in its knowledge base. This leads to the following research question.

Subquestion 4 How should we implement the reference ontology to build the operational ontology?

Constructing the operational ontology by implementing the reference ontology in a computer readable format, together with a demonstration that the operational ontology can answer the competency questions, concludes SABiO's processes. This leaves us with the final goal of integrating the ontology into Plasido.

Subquestion 5 How do we incorporate the explanation ontology into the Plasido engine?

The integration of several explanations into one explanation tree requires an algorithmic solution. Therefore, a sub-question of the software development problem is:

Subquestion 5.1 How do we combine several instances of the ontology from various sources to construct an explanation hierarchy?

1.4 Structure

In the second and third chapter we introduce the reader to the domain of explanation. The second chapter consists of both an overview of existing explanation algorithms, as well as of several attempts at a categorisation of those. Simultaneously, we pinpoint some requirements of the eventual ontology that the literature gives rise to. At the end of chapter 2 we have answered the first subquestion. In the third chapter we continue with the theory from philosophy and the social sciences on what a useful explanation is, together with a discussion to what extent we should strictly follow either theory. Therefore, chapter 3 answers subquestion 2. Speaking in processes of SAbiO, the second and third chapters perform the requirements elicitation. We collect our purpose, requirements, ontology modules, and competency questions in chapter 4, thereby answering subquestions 3.1 through 3.3 and paving the way for the construction of the reference ontology.

Having constructed the requirements in natural language allows us to introduce Description Logic, the formalism underlying ontologies, in the closely related chapters 5 and 6. We introduce syntax and semantics as far as necessary for our purposes, during which we construct the non-controversial basis of the eventual ontology. Chapter 5 does not answer a research question by itself, but defines the logic we use in chapter 6.

Research subquestion 3 is answered in chapter 6. The chapter starts with a discussion of previous attempts at an explanation ontology. The rest of chapter 6 concerns a formalisation of subquestion 3.1. The collection of Description Logic formulas constructed in the chapter answers research subquestion 3.

Having constructed the reference ontology allows us to operationalise the ontology in chapters 7 and 8. The seventh chapter makes the reader familiar with the Web Ontology Language (OWL), the W3C standard for ontologies. It answers subquestion 4 by showing the operationalisation of the reference ontology in OWL. Chapter 8 will conclude the ontology design process by showing that the competency questions (subquestion 3.2) can indeed be answered by the ontology.

We present our proof of concept in the penultimate chapter 9 to show that our extensions to the Plasido framework enable the integration of explanations in the format of the ontology. The general content of the chapter answers research question 5, while a specific subsection answers subquestion 5.1. In our final chapter we conclude with an answer to the research questions, some discussion, and directions for further research.

Chapter 2

The domain: knowledge base explanation

In this chapter we will introduce the domain of knowledge base explanation. We restrict our research to machine learning models, Bayesian Networks, and rule-based expert systems. The explanation of machine learning models was the cause of this research. We cover Bayesian networks, because it is another important learning paradigm which does not lead itself to intuitive explanation (Timmer et al. 2017). Rule-based expert systems conclude the list for three reasons. First, these systems were the first AI systems to be widely applied (DARPA 2016). Second, rule-based systems are typically judged to be easily interpretable (Guidotti et al. 2018). Third, the Plasido framework uses rules to define the hierarchy of knowledge bases. We will discuss the most relevant explanation methods for each type of KB.

In terms of SABiO, figure 2.1 gives a schematic view of the first phase. We will perform the roles of domain expert and ontology engineer. The role of ontology user will not be filled by a natural person, but we will be filled by the project description and by the TNO supervision. No concrete ontology user has been identified yet. The goal of chapter 2 and chapter 3 is to formulate the domain, with chapter 2 presenting the domain of KB explanation. Chapter 3 gives relevant lines of research from philosophy and the social sciences. Chapter 4 will draw the purpose, requirements, ontology modularisation, and competency questions from the domain study of chapters 2 and 3.

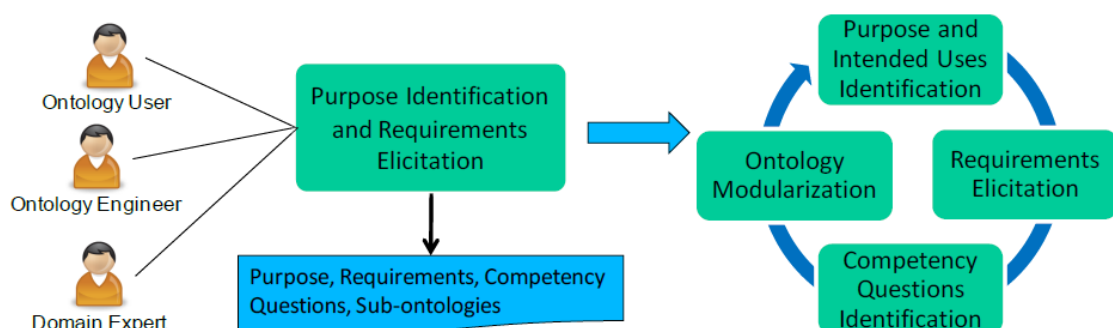


Figure 2.1: The first phase of SABiO (Falbo 2014)

2.1 Machine learning

Machine learning (ML) is typically divided into **supervised** learning, **reinforcement** learning, and **unsupervised** learning. A supervised learning algorithm takes as its input a set of data points with a target label. The algorithm “learns” a predictor such that it can predict the label of a previously unseen data point. Although the learning mechanism usually is probabilistic, the resulting predictor is deterministic. If there are no labels present, we call it unsupervised learning, because the algorithm does not get feedback. The algorithm resorts to a type of clustering. A reinforcement learner does get feedback, but not as labels. Each state can be input to a fitness function to check its value. A correct prediction is reinforced by an increase in the fitness function. In our research we will consider supervised learning, because that is the only type which is widely applied in practice.

Formally, we have a set of examples $D = \{z_1, z_2, \dots, z_n\}$, where each example is a pair $\langle \vec{x}, y \rangle$. The set of examples is divided into a large training set and a small test set. The input \vec{x} represents a single data point with y its target label. The supervised machine learning algorithm takes the training set as its input and learns a new function $f : X^N \rightarrow Y$ such that it can predict a class or value for another data point. The accuracy of the new function is tested by running it on the test set. The exact implementation depends on the particular algorithm, which can be sophisticated to the extent that the developer may wonder why the algorithm learned a particular function. Therefore, a machine learning algorithm is often called a black box. We will also use the term “model” to refer to the function learned by the machine learning algorithm.

In the rest of this section we will describe three taxonomies or categorisations of machine learning explanation algorithms. At the same time, we will already try to elicit some general concepts. Then, we will turn to a more conceptual discussion on what explainability or interpretability actually is.

2.1.1 Taxonomies

Biran and Cotton (2017) show the large variety of explanation algorithms in their overview paper, giving a bottom-up view of the field. We, however, will focus on the papers that use a top-down approach and that have provided a high-level view of the kinds of explanation algorithms in use. If the high-level categorisation is correct, fitting the ontology to capture all high-level concepts necessarily also captures the low-level concepts.

We will first describe the taxonomy by Guidotti et al. (2018), because they provide the most extensive categorisation yet. We will then compare it with the categorisations given in Gilpin et al. (2018) and Lipton (2016).

Guidotti et al.’s taxonomy

Guidotti et al. (2018) provide the most extensive and fleshed-out taxonomy of explanation algorithms, summarised in table 2.1. They propose a categorisation of explainability algorithms based on four properties: 1) the type of problem, 2) the unboxing algorithm, 3) the machine learning algorithm, and 4) the type of input.

The first category *type of problem* shows the different approaches to explainability.

Type of problem	Unboxing algorithm	Type of input	ML algorithm
Model explanation	Decision tree	Text	Neural network
Outcome explanation	Decision rules	Tabular	Tree ensemble
Model inspection	Feature importance	Images	Support vector machine
Transparent box	Salient mask		Deep neural network
	Sensitivity analysis		
	Partial dependence plot		
	Prototype selection		
	Neurons activation		

Table 2.1: Schematic view of Guidotti et al.'s taxonomy

On the one hand, we may have used a machine learning algorithm to construct a predictor, that, afterwards, we apply an explanation algorithm on to interpret it. On the other hand, we may construct a predictor that is by itself interpretable, and does not need an additional method to explain itself. In the former case we need to *reverse engineer* an ad-hoc solution to explain the model. In the latter case we face the problem of *designing* a transparent box instead of a black box.

A reverse engineering solution is more easily constructed than a transparent box, so is more widely applied. Therefore, Guidotti et al. subdivide the category of reverse engineering.

Model explanation: An additional KB is learned to mimic the behaviour of the black box, but with increased interpretability traded for a decrease in accuracy. The original classifier can be run to compute the output, while the proxy algorithm gives an approximation of its inner workings. We can interpret the complicated KB through inspection of the proxy.

Outcome explanation: An additional algorithm is developed that takes the machine learning model and a data point as input. It outputs an explanation for that particular data point.

Model inspection: A graphical or textual explanation is constructed to explain the black box as a whole.

The difference between model explanation and model inspection may need some elaboration. In the former case we end up with two classification functions. One is the original model trained for optimal performance. The other is an approximation of the original with increased interpretability due to the decrease in complexity. The model inspection problem does not need a second classifier, but may suffice with a graphical or textual explanation of the network.

Furthermore, Guidotti et al. identify eight unboxing algorithms to open a black box. We order them by the subtype of black box explanation (i.e. model explanation, outcome explanation, and model inspection) the particular type of explanation usually belongs to.

Model explanation

Decision tree: The machine learning model is probed with made-up input to test its behaviour. The explainability algorithm outputs a decision tree based on the model's behaviour, such that the decision tree is an approximation of the machine learning model. In other words, the explainability algorithm learns a simplified proxy of the machine learning model that is supposed to be more interpretable due to its lower complexity.

Decision rules: This kind of explanation algorithm is similar to the previous one. The machine learning model is tested through a series of dummy data. The resulting behaviour is modelled as a set of decision rules. We gain an increase in interpretability given the assumption that decision rules are inherently easier to interpret than a machine learning algorithm.

Outcome explanation

Feature importance: When introducing machine learning, we showed that it tries to find correlations between the several features of a data set. The *feature importance* explainability algorithm tries to find the features that were the most important for a given classification. The value of feature X may make no difference to the classification of a data point, when a change in the value of feature Y would change the classification. Therefore, feature Y is the more important feature. The algorithm may give several important features, not necessarily only the *most* important one. An example output is presented in figure 2.2.

Salience map: This explainability algorithm, which is also called a *salient mask*, applies a series of filters on top of the input to test which part elicits the strongest response. It is usually applied on explanation of image recognition. The algorithm shows which part of the input elicited a particular classification.

For example, figure 2.3 shows that the algorithm can identify which parts of the input gave rise to each of the three predicted classes: labrador, electric guitar, and acoustic guitar. Assuming the output of the algorithm is valid, the body of the guitar caused the classification as acoustic guitar, whereas the neck caused the classification as electric guitar. We may even infer that misclassification of an acoustic guitar as an electric guitar is more likely than vice versa. After all, both types have a similar neck, but a very different body. So, the salience map can explain why an incorrect prediction is nevertheless based on a correct inference.

Moreover, the algorithm can also help detect a correct classification based on unrelated features. Figure 2.4 shows that the image of the husky is correctly classified as a husky. However, the salience map algorithm shows that the snow in the background elicited a stronger response than the husky itself. It actually is a real problem in the field of image recognition that the classifier may base its classification on watermarks or the environment instead of the actual content. However, please note that the husky example was designed to illustrate this problem instead of an accidental failure.

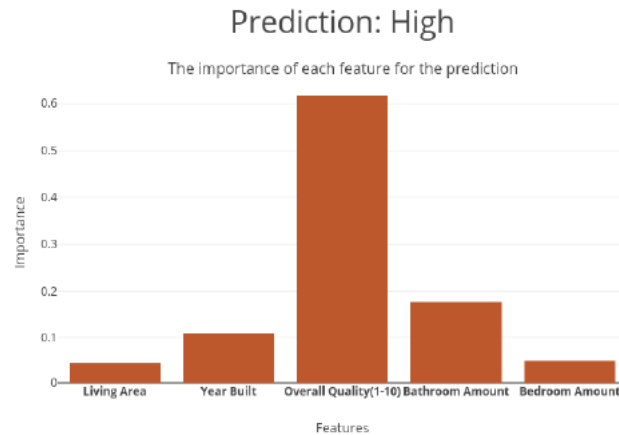


Figure 2.2: Example of feature importance output (Adhikari 2018)

Prototype selection: This algorithm tries to find an element from the training set that is most similar to the new input. Given the similarity, the prototype should explain why the requested input received the same classification. Clearly, this algorithm needs access to the training set *and* the machine learning model, while most other explanation algorithms do not.

Model inspection

Sensitivity analysis: This method analyses the correlation between the uncertainty in the output of the model and the uncertainty in the input.

Partial dependence plot: The method can visualise the correlation between an input feature and the output. This way, it can be shown whether the correlation is linear, monotonic, or more complex.

Neurons activation: Each node in each vertical layer outputs a certain value to the next layer. The *neurons activation* algorithm shows for which input a specific area of the network outputs higher values. This way we can compare inputs on the areas where a strong response is elicited. By manually comparing the content of the images, we may even observe which areas react to which stimuli. Clearly, this algorithm is only applicable to neural networks, while most other explanation algorithms are *model agnostic*, i.e. are applicable independent of the type of algorithm.

The division of unboxing algorithm into the three types of problems (i.e. model explanation, outcome explanation, and model inspection) is not as strict as presented above. A decision tree proxy may perform the role of second classifier in model explanation, but may also be employed to compute a single path to explain a particular data point.

Guidotti et al. also consider four machine learning models: 1) neural network, 2) tree ensemble, 3) support vector machine, and 4) deep neural network. The last feature, type of input, is divided into 1) text, 2) images, and 3) tables where each feature is either

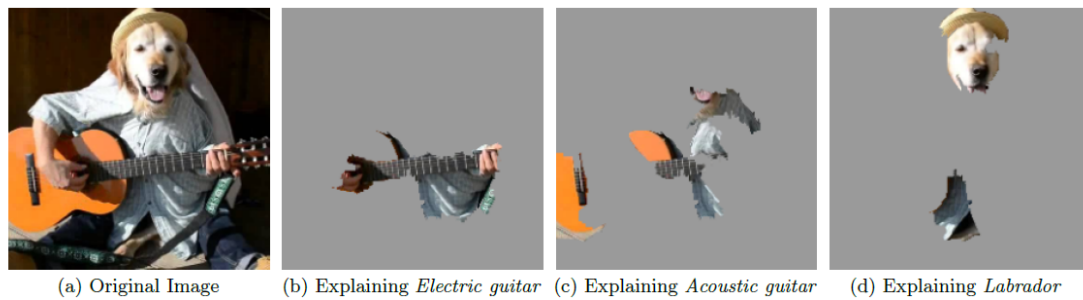


Figure 2.3: This example from Ribeiro et al. (2016) illustrates the application of a saliency map to find why the picture was classified as showing a labrador, an electric guitar, and an acoustic guitar.

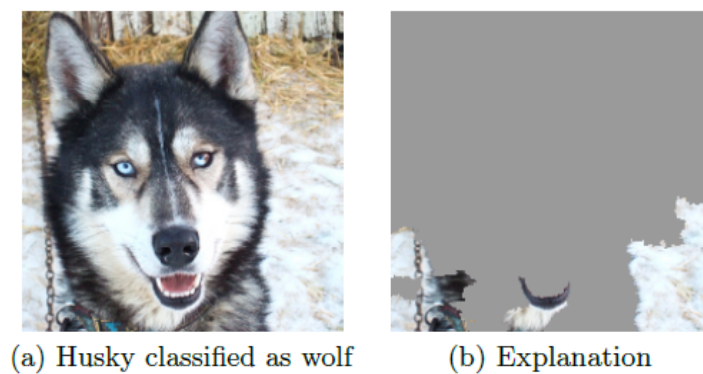


Figure 2.4: This example from Ribeiro et al. (2016) shows how a saliency map can help detect that the classifier bases its prediction on the wrong features.

numerical, boolean, or categorical. This difference is not very relevant for our research, because the type of data has little influence on the form of the explanation.

We would like to note that the categorisation of explainability algorithms could have been more fleshed out. Guidotti et al. categorised the types of explanation independently from the explanation extraction method. Their categorisation does not show which explanation extraction method functions as which explanation type. They described in natural language that, for example, both the *decision tree* and the *decision rules* algorithms can function as both model explanation and outcome explanation. In the case of model explanation, a decision tree algorithm would construct a tree simplifying the knowledge base. In the case of outcome explanation, the decision tree algorithm would construct a path explaining the outcome. On the other hand, the methods we described as outcome explanation methods (i.e. feature importance, salience map, and prototype selection) function necessarily as outcome explanation. Guidotti et al.'s category does not contain this difference, but the authors explain it in the accompanying paper. Our ontology should be computer readable, so cannot have an accompanying natural language explanation. Therefore, we want the differences between the various explanation extraction methods to be inherent to the ontology.

Concluding, Guidotti et al. argue that the type of problem, unboxing algorithm, supervised learning algorithm, and type of input together characterise an explainability case. For our purposes, their identification of the unboxing algorithms is the most important, since that part decides the contents of the explanation that have to be captured in the ontology. However, we noted that we need a categorisation of explanation algorithms that is more fleshed out than the one Guidotti et al. provides, although their attempt does provide a starting point. Therefore, in the next section we will compare Guidotti et al.'s work with Gilpin et al. (2018).

Gilpin et al.'s taxonomy

Gilpin et al. (2018) give a categorisation of methods to explain the processing of a Deep Neural Network, i.e. a neural network with more than one hidden layer. Our definition of a KB is broader, but the methods described by Gilpin et al. are *model-agnostic*. In other words, they are independent of the machine learning model and not exclusively suited to neural networks.

Gilpin et al. consider *interpretability* to be distinct from *explainability*. They first provide a categorisation of interpretable models. They then provide methods they consider actual explainability algorithms. Their categorisation of interpretability algorithms is not unlike the categories Guidotti et al. (2018) provided for unboxing algorithms.

Linear proxy methods: As with all proxy methods, a simplification of the machine learning model is learned. Linear proxy methods belong to the model explanation category of Guidotti et al.'s categorisation, although the method can be adapted to function as outcome explanation (Ribeiro et al. 2016). A linear function is supposed to be easily interpretable compared to a neural network.

Decision trees: Explaining a model through a simplification modelled as a decision tree is also an element of Guidotti et al.'s categorisation, where it can belong to either the model explanation category or the outcome explanation category. Gilpin et al. only consider the decision tree method for model explanation.

Saliency mapping: Using a saliency map to show which parts elicit the highest response is also an element of Guidotti et al.'s categorisation. This category necessarily belongs to Guidotti et al.'s outcome explanation category.

Automatic rule extraction: The fourth element of Gilpin et al.'s list occurs also in Guidotti et al.'s categorisation as "decision rules". It depends on the implementation whether automatic rule extraction belongs to model explanation or to outcome explanation.

Gilpin et al. also provide three types of *explanation-producing systems*. These systems are explicitly designed to be more explainable, while the interpretability algorithms are applied in an ad-hoc fashion to models that are otherwise not interpretable at all. In Gilpin et al.'s terminology these systems have advanced beyond mere interpretability to be explainable.

Explicit attention: These systems use different techniques on different parts of the input. For example, when classifying birds one layer may respond to features of the body, whereas another may respond to features in the head of the bird (Xiao et al. 2015).

Disentangled representation: This method tries to distinguish between neurons that were meaningful from neurons that are independent of the classification.

Generated explanations: These systems generate a syntactically correct sentence that is shown to the user. First, an explanation is generated, which is then put into a sentence containing "because". The language parts of the network are usually trained on large sets of human written explanations.

However, the work on explainable models is still in its infancy compared to the work on interpretable models.

In summary, Gilpin et al. find four kinds of interpretability algorithms of which three are also explicitly mentioned in Guidotti et al.'s list of unboxing algorithms. The fourth entry is the linear proxy method, which belongs to the model explanation category. Gilpin et al. also mention three types of explainability algorithms, all of which do not occur in Guidotti et al. When discussing Guidotti et al. (2018) we noted that the categorisation could be more precise, since we found similarities that were not reflected in the categorisation. This is also true for Gilpin et al. Their list of interpretability methods contains three types of proxy methods. So, we should change the categorisation to two high level categories: 1) proxy methods and 2) saliency mapping. Explanation by linear proxy, decision trees or decision rules become subtypes of the supertype "proxy methods".

Lipton's taxonomy

The third and last categorisation we consider follows Lipton (2016). It can hardly be considered a taxonomy given its small size. Lipton observes five types of interpretability algorithms, all subtypes of black box opening algorithms rather than transparent box design algorithms. Lipton prefers the term *post-hoc interpretability* for black box opening algorithms. We consider his categories succinctly, because several overlap with categories already described.

Text explanations: This category captures all explanations outputted in a pure text format. It does not have a corresponding category in Guidotti et al.'s taxonomy, but could fulfil the role of both outcome explanation and model inspection. It fits well into Gilpin et al.'s generated explanations category, even though Lipton considers it a post-hoc interpretation rather than a transparent design.

Visualisation: All methods that perform their explanation by presenting a set of visualisations of the whole model belong to this category. Depending on the implementation it can be a sensitivity analysis, a partial dependence plot from Guidotti et al. or another implementation of model inspection. It does not have a corresponding element in Gilpin et al.

Local explanation: Lipton considers salience mapping a prime example of local explanation. Although it uses images, it does not belong to the visualisations category as it does not visualise the complete network. Rather, the salience map shows the focus of the model on one particular input only. The salience mapping method has a corresponding entry in both other taxonomies.

Another type of local explanation is LIME (Ribeiro et al. 2016), where a linear model is learned on a subset around a particular point. This linear model is likely to be simulatable, since it only aims to approximate the complete model on a subset of the data. This type of local explanation has corresponding elements in both other taxonomies.

Explanation by example: This category includes all explanations where a prototype is presented. It corresponds to Guidotti et al.'s Prototype selection, but has no corresponding element in Gilpin et al.

2.1.2 Extracting concepts

Having described three taxonomies, we will turn to an even more conceptual level. Several researchers argue that the field of explainable ML cannot progress beyond its current state if the field does not agree on a set of definitions for explainability (Guidotti et al. 2018; Doshi-Velez and Kim 2017; Gilpin et al. 2018; Lipton 2016). Most papers simply propose an explanation algorithm without defining *a priori* what exactly constitutes an explanation.

Explanation versus interpretation

On the highest level, even the definitions of *explanation* and *interpretation* are not agreed upon. Gilpin et al. state that “explainable models are interpretable by default”, but do not think the reverse holds. They consider interpretability merely as a first step consisting of, for example, a salience map or a simplified proxy. True explainability is only reached by KBs that can, for example, produce insight into the causes of their classifications, can provide meaningful responses to why-questions, or can be audited.

Not all researchers agree with this distinction. Doshi-Velez and Kim (2017) and Guidotti et al. (2018) define a model to be interpretable if it has the ability to explain its mechanisms to a human. They equate interpretability with explainability. Miller (2019), observing the problem from the viewpoint of the social sciences, also uses interpretability

as a synonym for explainability. He differentiates between explainability (whether a human can understand a model) and an explanation, which is the by-product of the classification intended for the user. We will give an operationalisation of interpretability in the next section.

Operationalisation of interpretability

In Guidotti et al.'s taxonomy we already noted the difference between the explanation of an outcome and the explanation of the model. Constructing a simplified proxy to explain a complex KB involves an important trade-off between *interpretability* and *accuracy* or *completeness*. A decision tree, rule base, or linear model is often recognised as interpretable (Guidotti et al. 2018, p. 8), so is also considered suitable to fulfil the goal of learning an intelligible proxy. The goal of accuracy is to present the most accurate explanation of the KB. Lipton (2016) divides interpretability into three parts:

Simulatable: Simulatability is transparency on the level of the whole model. Only if a human subject can reproduce the computations performed by the actual model, will we call it simulatable. For example, a decision tree with few sub-branches is most likely simulatable, but doubling the number of nodes will cause it to quickly lose the property.

Decomposability: We will call a model *decomposable* if it is understandable on the level of individual features. The performance of inflexible classifiers, like linear models, can often be increased by preprocessing the variables. A linear model can, as the name suggests, only learn a line in two dimensions or a plane in three dimensions. Transforming the data space by introducing new variables as the aggregate of some original variables often increases the performance. While the original features of a data set are usually trivially interpretable, that may not be the case for custom features. So, introducing preprocessed features will very likely cause the model to lose the decomposability property.

Algorithmic transparency: On the lowest level, Lipton identifies the *algorithmic transparency* property. A model satisfies this property if we can in some way visualise the learning process. According to Lipton, decision trees, rule sets, and linear models have the algorithmic transparency property. After all, we can easily visualise the construction of a decision tree or the transformations of a linear curve for each added sample. These models are sometimes generally regarded as “easily understandable and interpretable for humans” (Guidotti et al. 2018). Therefore, the extra properties of simulatability and decomposability are especially helpful to show the existence of specific linear models, decision trees, and rule sets that are not easily interpretable for humans.

The goal of *accuracy* can be trivially satisfied by presenting all weights and parameters of a system. This, of course, is not interpretable for humans. The more an explanation is a simplification of the underlying KB, the less accurate an explanation it is.

This division of interpretability leads us to doubt the assertion that decision trees, rules, and linear models are inherently interpretable. These models only satisfy the *algorithmic transparency* property. They may be decomposable and probably are not

Category	Variable	Values
Content	Focus	evidence / model / reasoning
	Purpose	description / comprehension
	Level	micro / macro
	Causality	causal / non-causal
Communication	User-system interaction	menu / predefined questions / natural language dialogue
	Display of explanations	text / graphics / multimedia
	Expressions of probability	numeric / linguistic / both
Adaptation	User's knowledge about the domain	no model / scale / dynamic model
	User's knowledge about the reasoning method	no model / scale / dynamic model
	Level of detail	fixed / threshold / auto

Table 2.2: Categorisation of explanation methods following Lacave and Díez (2002)

simulatable, even though simulatability is most needed to perform the role of an interpretable proxy. Most machine learning models are very complex, such that a proxy will either be too inaccurate to be a reasonable approximation or too complex to be simulatable (Lipton 2016).

2.2 Explanation in other knowledge bases

Having described the field of machine learning explanation, we turn to Bayesian networks (BN) and expert systems. Lacave and Díez (2002) present a survey of explanation in BNs. They describe each type of explanation along the categories shown in table 2.2. In Lacave and Díez (2004), they applied the same approach to explanation of expert systems. Their approach was broader than ours, since they included probabilistic expert systems, whereas we do not. It is already promising to see a single categorisation applied on both BNs and expert systems. In the rest of this section we will first describe rule-based expert systems and Bayesian networks. We will then take a look at Lacave and Díez's taxonomy and at two explanation methods for BNs: Elvira (Lacave et al. 2007) and the Two-phase Method (Timmer et al. 2017).

2.2.1 Rule-based expert systems

A rule-based expert system consists of a knowledge base and an inference engine. The knowledge base is divided into facts and rules. The facts represent the initial knowledge of the system. The rules $(p_0, \dots, p_n \rightarrow q)$ allow the inference of new facts. The reasoning engine is an algorithm that searches for new facts that can be derived from the knowledge base. As a means of explanation, the reasoning engine can track which rules it followed to derive a new fact. If it fails to derive a particular fact, the reasoning engine may return a collection of facts that would allow the inference of the new fact.

2.2.2 Bayesian networks

A Bayesian network (BN) is an acyclic directed graph, where the nodes represent variables and the edges represent probabilistic dependencies. For each node the probability is given of it being true, which together with the probability of being false should add up to 1. These probabilities are given for every configuration of parent nodes. So, in example 2.5 the node *Rain* needs only a probability for the chance of there being rain. However, the *Sprinkler* node is dependent on its parent, so we should give the probability of the sprinkler being on in both the case of rain and without rain. The node *Grass wet* has two parents, so its parents can have four true-false configurations. The joint probability can be computed through

$$Pr(G, S, R) = Pr(G|S, R)Pr(S|R)Pr(R)$$

where G , S and R stand for the nodes *Grass wet*, *Sprinkler*, and *Rain* respectively. As an example we compute the case where the grass is wet, the sprinkler is turned on, and it is raining:

$$\begin{aligned} Pr(G = T, S = T, R = T) &= Pr(G = T|S = T, R = T)Pr(S = T|R = T)Pr(R = T) \\ &= 0.99 \times 0.01 \times 0.2 \\ &= 0.00198 \end{aligned}$$

However, in most applications of Bayesian networks we are not interested in the joint probability, since that requires us to know the value of each variable. Instead, we usually only know the value of a proper subset. For example, we would like to know the chance of rain when we have only observed that the grass is wet. In other words, we want to compute the conditional probability that it is raining given that the grass is wet. We use the conditional probability function:

$$Pr(X|Y) = \frac{Pr(X \cap Y)}{Pr(Y)}$$

In this case we instantiate the variables as:

$$Pr(R = T|G = T) = \frac{Pr(G = T, R = T)}{Pr(G = T)}$$

This computation may still be intuitive, but computations on larger networks may become complex, such that they are not understandable. Therefore, several methods have been developed to explain the computation. This explanation concerns two aspects. First, the flow of the probabilities through the network can be visualised to show via which ancestor nodes the probability flowed to the end node. Second, we may show how to interpret the exact numerical value outputted by the algorithm to explain the difference between a probability of $\frac{1}{10.000}$ and of $\frac{1}{1.000.000}$.

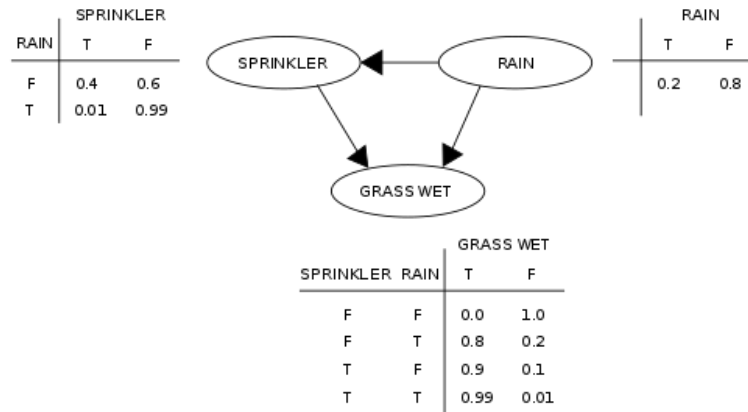


Figure 2.5: Example Bayesian network

Taxonomy by Lacave and Díez

Table 2.2 shows the categories used by Lacave and Díez to characterise both Bayesian network explanation (Lacave and Díez 2002) as well as expert system explanation (Lacave and Díez 2004). It is divided into aspects concerning 1) the content of the explanation, 2) the communication between user and system, and 3) the real-time adaptation of the explanation to the user.

The first variable is *Focus*, having three possible values: 1) evidence, 2) model, and 3) reasoning. The authors call a focus on explaining the evidence *abduction*, which is reasoning from an observation backwards to the most likely explanation. Then, the division of either explaining the model or explaining the reasoning closely mirrors the division in XAI between model explanation and outcome explanation.

The categorisations concerning communication between user and system, and the adaptation of the system’s explanation to the user, are not very useful to our purposes. We admit that the way an explanation is offered to the user is “crucial” (Lacave and Díez 2002), but the presentation is usually independent from the way the explanation is extracted from the network. Only the *generated explanations* from Gilpin et al. (2018)’s taxonomy springs to mind as an exceptions. However, our aim is to build an ontology of *explanation*, rather than an ontology of *explanation communication*.

Elvira

Elvira (Lacave et al. 2007) is a program for the explanation of Bayesian networks. It performs the explanation on three levels. First, Elvira can give an explanation on the level of individual nodes. Figure 2.6 shows nodes Disease 1, Disease 2, Virus A, and Virus B in the *expanded* mode, compared to the other nodes which are, at the moment, *contracted*. The expanded view shows the prior and posterior odds. Second, the graphical view automatically gives the links different colours based on the effect the link has. Edges representing a positive link, where the occurrence of the parent increases chances that the child occurs, are shown red, while negative links are coloured blue. Moreover, the link becomes wider the stronger the association between the two variables is. Third,

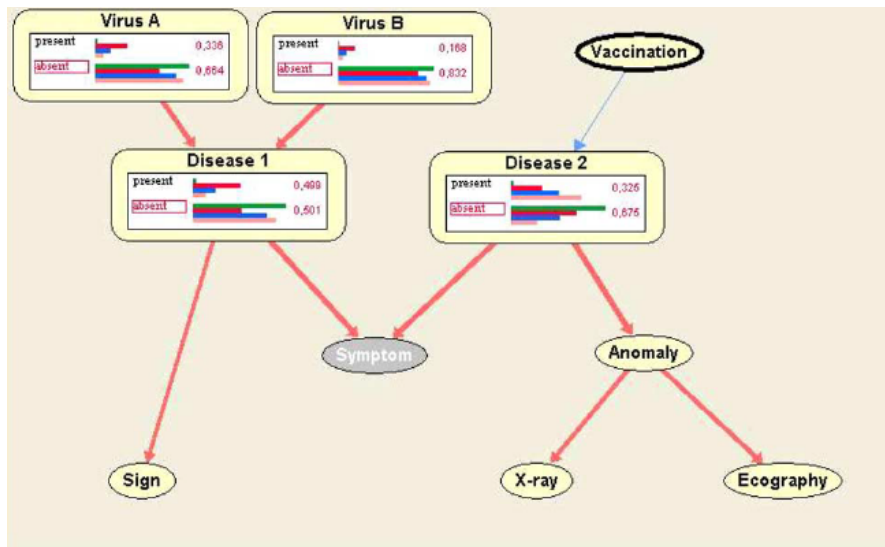


Figure 2.6: Example Elvira diagram (Lacave et al. 2007)

the network can output a natural language explanation summarising the explanation in the nodes and in the links. For example, the network in figure 2.6 may be explained as “The network represents the following information: The disease/anomaly Virus A has neither causes nor risk factors represented in the network. It may cause the following: DISEASES/ANOMALIES: Disease 1, SYMPTOMS: Symptom, SIGNS: Sign.” (Lacave et al. 2007).

Moreover, the expanded node in figure 2.6 shows four bars for coloured bars for both the case where the variable is present and where the variable is absent. The first bar, coloured in green, represents the a priori probability of each node. The other three bars represent manually added use cases. In a specific use case, the observed variables have been entered, allowing the the network to compute the conditional probabilities of the other nodes.

Timmer et al.

The method by Timmer et al. (2017) is based on argumentation theory. Whereas Lacave et al. explain a BN through visualisations and textual explanation, Timmer et al. transform the BN into a more intuitive structure. After all, the links in the BN represent (causal) dependence, which may be either positive or negative. Timmer et al. elicit an argumentation structure from the network to intuitively show which facts support other facts. Figure 2.7 shows the support graph, which is the first step in their two-phase transformation method.

The support graph performs the role of outcome explanation. Given a set of variable instantiations we can construct a specification of the support graph to explain the value of the output variable. Figure 2.8 shows an instantiation with a set of observed variables, identifiable through the double borders. For the unobserved variables the expected value together with the likelihood ratio measure of strength indicates the increase in strength of a belief given the observed variables. Another option is to have these values reflect

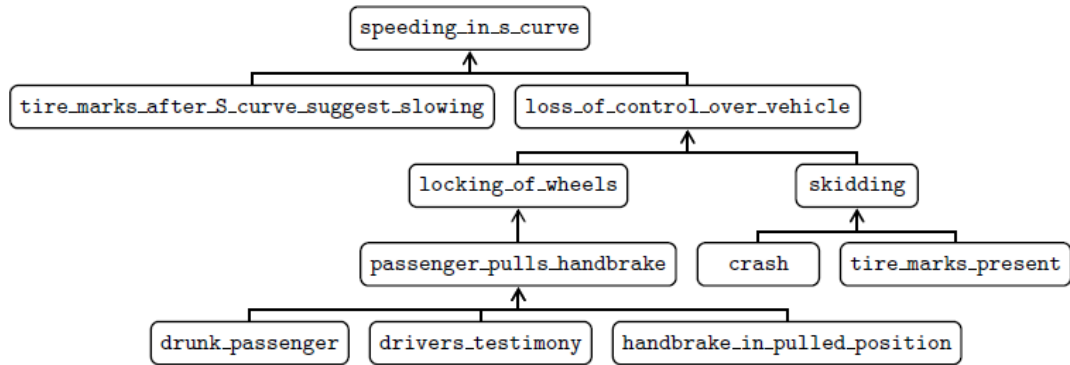


Figure 2.7: Support graph of a Bayesian network (Timmer et al. 2017)

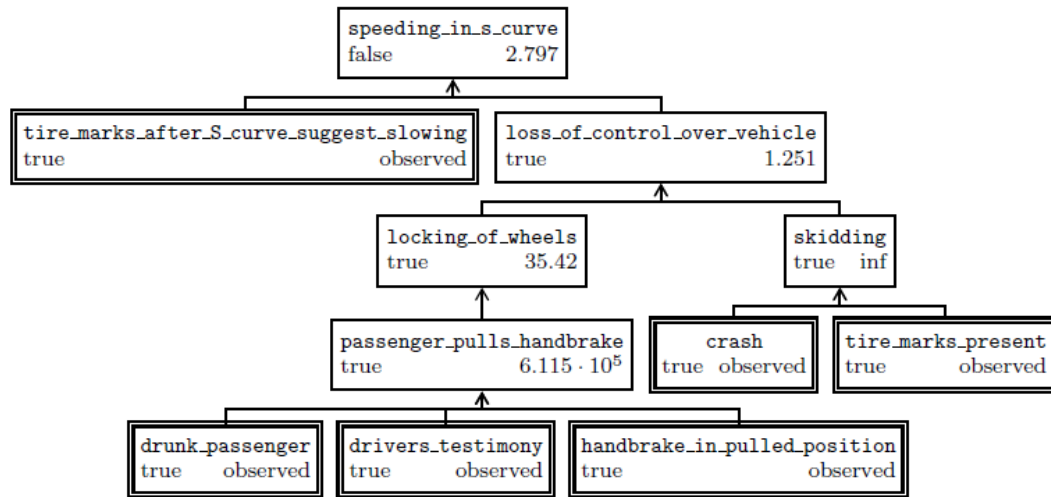


Figure 2.8: Support graph instantiated with a set of observed variables (Timmer et al. 2017)

the posterior odds, which gives an absolute instead of a relative measure of strength.

Summary

Summarising this chapter we find the categorisation by Guidotti et al. (2018) to be best suited for our purposes. First of all, it has already progressed beyond a simple categorisation towards a taxonomy. Second, the taxonomies by Lipton and Gilpin et al. are subsumed by Guidotti et al. Third, we can also capture explanation methods for Bayesian networks in their taxonomy, showing that the taxonomy is applicable beyond the domain of explainable machine learning. This leaves us to improve upon Guidotti et al. by incorporating their natural language remarks in the ontology. After all, the ontology is intended as a standard for communication by computers, so all information

has to captured by the ontology.

Chapter 3

Insights from the social sciences and philosophy

In this chapter we will discuss approaches from the social sciences and philosophy, along with their relevance to both our research and to KB explanation in general. Following philosophical terminology we will divide an explanation into an *explanandum* (plural *explananda*) and an *explanans* (plural *explanantia*). The former is the part that has to be explained¹. The latter refers to the part that does the actual explaining.

A large body of literature concerning explanation already exists in philosophy and the social sciences, but a definite theory is nowhere to be found (Miller 2019; Strevens 2006). Both fields have quite different aims when defining explanation. Definitions of explanation in the philosophy of science try to *prescribe* necessary and sufficient conditions for scientific explanation. The social sciences view explanation as a social act between at least two agents, and *describe* the types of explanations that result in a situation where the social interaction ends with the explanation having caused an increase in knowledge for at least one agent.

3.1 Insights from the social sciences

An explanation is often seen as a conversation between agents: “Someone *explains* something *to* someone” (Hilton 1990). Miller (2019) gives a survey of the social science where it is relevant for XAI research. He wrote the paper after Miller et al. (2017) found most XAI papers to be lacking in references to publications from the social sciences. Miller draws the following four conclusions:

Explanations are *contrastive*: Asking “Why *P*?” is a shorthand for “Why *P* rather than *Q*?”. Taken literally, the former question asks for the complete history that led to event *P*. The latter only asks for a very particular difference in the causal history of the observed event *P* compared to the expected event *Q*. We call the observed state *P* the *Fact* and expected state *Q* the *Foil*. In conversations the foil often is implicit.

We did not find contrastive explanation mentioned in the KB explanation survey,

¹Compare it with *promovendus* as “he who should obtain his PhD”.

despite the intuitiveness of the type of explanation. Therefore, Miller took it upon himself to formalise contrastive explanation for the advent of computational models (Miller 2018). Some of our TNO colleagues are among the few who constructed an algorithm for contrastive explanation (van der Waa et al. 2018), showing that the potential is acknowledged.

Explanations are *selected*: Humans are adept at selecting one immediate cause among a set of relevant factors. The person receiving the explanation can extract the part he or she judges to be the immediate cause, even when presented with a long string of information. This conclusion implies that KB explanation algorithms are not required to present one particular fact as explanation, but may present a collection of facts among which the user can select the immediate cause.

Probabilities do not matter: A causal explanation is usually more satisfying to humans than a mere statistical relation. Referring to a probability in itself does not say very much. This is already reflected in Elvira (section 2.2.2) and the Two-phase Method (section 2.2.2), where the individual probabilities of the network are but a small part of the complete explanation. Our survey of ML models also did not include a method where probabilities play a large role.

Explanations are *social*: An explanation takes place in a conversation between agents, where knowledge is transferred between both. The previous knowledge of the agents influences what kind of explanation is helpful. In our survey of KB explanation we found that systems responsive to the user's knowledge have not been developed yet. Lacave and Díez also included conversational explanation in their taxonomy, but found no explanation method that uses it to a sufficient degree.

3.2 Philosophical prescriptions

Whereas social theory argues an explanation can be constructed by selecting an immediate cause among a set of events, proponents of the *deductive-nomological model* (Lat. *nomos* = law) are suspicious of causality. They consider it to lack a “precise construal” (Strevens 2006) and therefore consider it not easily formalised. They argue that a scientific explanation should consist of a set of facts, among which is at least one generally valid law (Strevens 2006). Moreover, we should be able to arrive at the conclusion by consequently applying the generally valid law(s). For example, we should regard the classical example of *modus ponens* as a valid explanation of Socrates' mortality, because any valid inference is seen as a valid explanation for the conclusion.

Socrates is a human.
 Humans are mortal.
 =====
 Socrates is mortal.

The syllogism provides humans with a clear explanation of Socrates' mortality, because the explanation follows causal relation. However, the explanation following the causal direction is not a restriction placed upon the theory, but merely a property of the

particular example. The DN-theory has received quite some criticism, because it allows explanations that do not provide a causal relation (Lipton 1991).

For example, we may ask why the shadow of a flagpost measures five meter. The combination of the fact that the flagpost is ten meter high, and the rule that the length of the shadow of an object is half its length at a particular time of day, would be a valid explanation. Using the rule we can deduce that the length of the shadow should indeed be five meter. However, the general rule can also be stated the other way around:

the length of an object is double the length of its shadow.

Then we can use the fact that the shadow of the flagpost is five meter in length to construct the following explanation for the length of the original flagpost:

The shadow of the flagpost measures five meter.
The height of a flagpost is double the length of its shadow.

=====

The height of the flagpost is ten meters.

The major difference is, of course, that the original rule describes a causal relation, whereas the converse does not. However, the DN-model does not discriminate between both explanations. Common sense would dictate us that an explanation with causal relevance is more useful than one without.

The second cause of the weakness is that the DN-model does not impose a restriction on the generality of the rules and facts. The following explanation is typically used to illustrate this objection (Strevens 2006):

This salt has been hexed.
All hexed salt dissolves in water.

=====

This salt dissolves in water.

Although the rule *does* describe a causal relation, it is not considered to have explanatory value. It is too specific, since the more general rule “all salt dissolves in water” is equally applicable. We should consider an explanation to be better the more general its rules and facts are. Again, the DN-model does not impose any restrictions or orderings.

As a consequence of these objections a re-evaluation of causality emerged, resulting in *contrastive explanation* as an operationalisation of causal explanation (Strevens 2006). The theory assumes that every question is implicitly contrastive (Miller 2019; Lipton 1991). This way, we find a correspondence between the philosophy of explanation and the social sciences. The philosophical account *prescribes* when a contrast is a scientific explanation, which is done in a more sophisticated way than just calculating the difference in causal history between fact and foil (Lipton 1990; Lipton 1991). However, the correspondence between philosophy and social theory shows that some types of explanation can be both rigid and intuitive.

Still, we would like to give a disclaimer. We agree that the flagpost example provides a strong philosophical objection against the deductive-nomological account, but our aim is quite different from the philosophical goal. We need to provide a framework that describes the structure of an explanation. The lofty philosophical goal is to construct both necessary and sufficient conditions that define a satisfactory explanation, but our

framework should also be able to capture insufficient explanations. First, it is up to the operator to review the explanation on its explanatory value. It is not within our scope to build an ordering on the quality of an explanation, though this may be an interesting future extension. Second, our research is restricted to three types of KBs. Our categorisation is not intended to cover each type of insightful explanation a human might give. Especially in the case of expert systems, an explanation consisting of facts and rules may be the best we can get, since that is exactly what an expert system consists of. Therefore, we need to construct our ontology such that it can capture a rule-based explanation. Also, the prominence of contrastive explanation in both philosophy and the social sciences, together with its emergence in XAI, gives us reason to include it in our framework.

Summary

In this section we covered how philosophy prescribes an explanation and how social theory describes an explanation. We found a remarkable overlap in the form of contrastive explanation: explaining a fact by pointing at the differences from an expected foil. It is prominent in philosophy as an operationalisation of causation. The difference in causal history between fact and foil can be considered a cause of the fact. In social theory a contrast between fact and foil is observed in any “why”-question, even if not explicitly stated. Its prominence in both philosophy and social theory gives us reason to include contrastive explanation in our framework, even though an explanation algorithm using contrastive explanations seems to be lacking.

Chapter 4

Towards a reference ontology

In this chapter we will conclude the first phase of SABiO (figure 4.1) by drawing the purpose, requirements, ontology modularisation, and competency questions from the previous chapters. The purpose has not changed since the introduction. So, we will only summarise the purpose and goals. The requirements we will draw from the main findings of chapters 2 and 3. The ontology modularisation and the competency questions will guide our way forward by giving a preliminary structure of the ontology, and by giving the questions the ontology should answer.

4.1 Purpose and intended use

The purpose and intended use has not changed since our exposition in the introduction. The ontology is intended to capture the kinds of explanations typically extracted from KBs, and to combine the set of explanations into a hierarchical structure. The structure of the explanation also needs to capture the dependencies of the federated knowledge base from which the knowledge is drawn.

The ontology is intended to be employed in the Plasido knowledge engine, which is based on the semantic technology of Apache Jena. Plasido performs all communication to and from the independent KBs using a set of rules. Facts that are derived using an external KB are added to Plasido's own symbolic knowledge base. This set of rules is run by a reasoning engine that traces which combination of facts and a rule were used

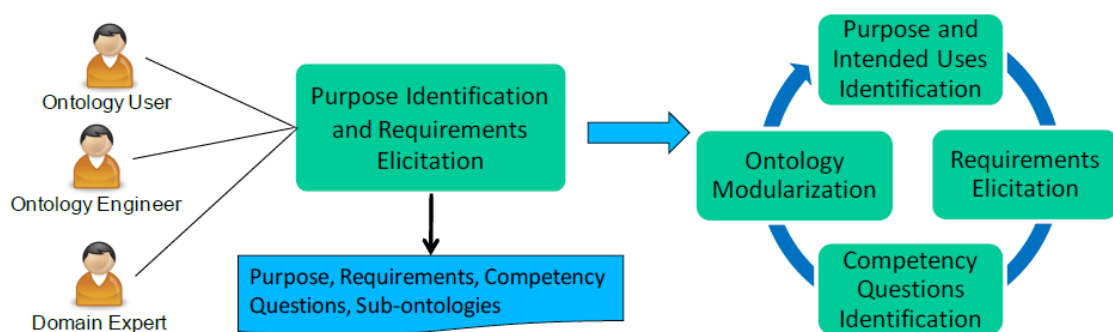


Figure 4.1: The first phase of SABiO again

to derive a particular conclusion.

Moreover, the explanation structure that is eventually outputted by the federated knowledge environment should be in the ontology format. The strict ontology definition allows the construction of queries that transform the ontology to a simplification containing only the requested data. An ontology is unreadable to non-professionals, but even professionals use queries to consult an ontology instead of manually inspecting it.

4.2 Requirements

In chapter 2 we gave an overview of the knowledge base explanation domain. The ontology should at least capture all structures that the explanation methods may produce. We identify five different types: 1) tree-based, 2) rule-based, 3) a set of items, 4) graphical, and 5) textual. A graphical or textual explanation has little internal structure, compared to a tree-based or rule-based explanation. Since we aim to provide a standard for communication that is used in practice, the class for both tree-based and rule-based explanation should have their internal structure fleshed out.

We also found that the structure of the explanation cannot be mapped one-on-one to the type of explanation. A tree-based structure can play the role of both model explanation as well as outcome explanation. In the former case we obtain a decision tree summarising the ML model. In the latter case we may obtain a decision tree that summarises a local sample of the data set. In the latter case we may also obtain a tree path for a single outcome. We use the same tree-based structure in all three examples. So, the ontology should have independent concepts for the structure and the four types of explanation: 1) model explanation, 2) outcome explanation, 3) model inspection, and 4) transparent box.

We also drew various KB types from the domain description. Especially the distinction between ML model, expert system, and Bayesian network is important for our purposes. We found that several KBs output an explanation of the same structure, so the KB that conceptualised the explanation should be part of the ontology. Otherwise, the difference between a rule-based proxy and a rule-based expert system would be lost.

The philosophical and social background showed us that an explanation can explain not just one particular fact. It can also explain the contrast between an observed fact and an expected foil. So, our ontology should model the difference between a contrastive explanation and a simple non-contrastive explanation.

Finally, from our purpose and our goals we draw the requirement that the ontology should capture a hierarchy of explanations. One explanation functions as the root, while other explanations merely support the root explanation. A tree-based structure is the most logical structure for the explanation hierarchy.

4.3 Ontology modularisation

We will first give our proposed ontology modularisation before giving the competency questions, so we can order the competency questions by module. In figure 4.2 we give our proposed modularisation, consisting of three modules.

Explanation of a fact: This module concerns the explanation of a single fact. It

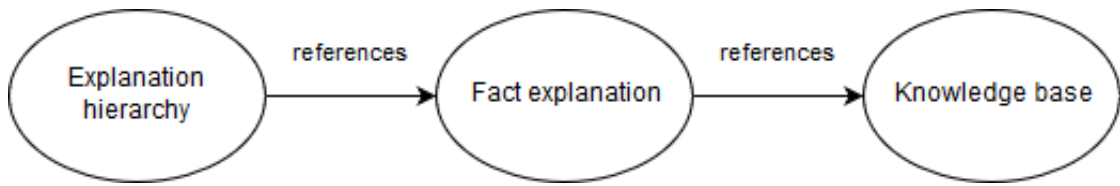


Figure 4.2: Modularisation of the ontology

should distinguish between the fact being explained (explanandum) and the structure that performs the actual explanation (explanans). The modelling of the explanans should capture the structures that emerged from the literature survey as we described in the previous section.

The internal structure of an ontology is a set of triples. So, a fact constitutes the combination of a subject, a predicate, and an object. The concrete representation of the ontology will be made more explicit in chapter 7.

Explanation hierarchy: Having defined an ontology for the explanation of a single fact, the next step is to link these facts into a hierarchy. This module needs to define the structure of a tree such that it can construct a hierarchy of explanations. The nodes of the tree should be explanations of single facts. The edges should connect a fact performing the explanation to its own explanation.

For example, a rule-based explanation may consist of a rule and three facts which together explain a conclusion. The rule and facts themselves may also have an explanation.

Knowledge base generalisation: This module will give the subset relations among the types of KBs. It needs to specify which subtypes exist of particularly ML algorithms, but should also include relevant sub-types of expert systems and BNs. We will not make this ontology module more precise than is required by the use case. The open world nature of ontologies allows future use cases to further specify this module if needed.

4.4 Competency questions

The competency questions play a double role in the SABiO procedure. First, they force us to define in natural language the information we want to be able to extract from the eventual ontology. The formal implementation of the competency questions (chapter 8) proves that the ontology can indeed answer these natural language questions. Second, the implementation of the competency questions allows us to inspect the ontology. The concrete representation of an ontology usually is unintelligible for both experts as well as non-professionals. The competency questions extract the required facts from a large collection of data.

We divide the competency questions into the three ontology modules.

4.4.1 Fact explanation

The following competency questions are for the explanation of a single explanation. It is mainly intended to extract information on the fact being explained, the structure that does the actual explaining, and the KB that produced the fact.

User queries about a single explanation:

1. General:
 - (a) What fact is being explained?
 - (b) What is the function of the explanation? Does it simplify the model, explain the outcome, or inspect the model?
 - (c) Which items does the explanation consist of?
 - (d) Which type of KB produced the fact being explained?
2. Model explanation:
 - (a) What is the proxy model?
 - (b) What type of KB is the proxy model?
3. Outcome explanation:
 - (a) What are the contents of the explanans?
 - (b) Which type of outcome explanation is the explanans an instance of?
4. Model inspection:
 - (a) What are the graphical or textual explanations?
 - (b) Is it a graphical explanation? Is it a textual explanation?

We can generalise the questions for model explanation, outcome explanation, and model inspection. The first question asks the contents of the explanans. The second question asks the subtype of the explanans. We aim to generalise the construction of the ontology such that one formalised query can draw this information from the explanans independent of the type of explanans.

4.4.2 Explanation hierarchy

The following competency questions are for an explanation tree. These questions concern remodelling the tree to extract a subset of the information contained in it. After all, the explanation tree contains information on the explanandum, explanans, and KB of each individual explanation. Therefore, we construct three competency questions, with one extracting the explananda, another giving the KB hierarchy, and a third showing the explanantia. These three competency questions together show that the explanation tree contains complete information of its individual explanations. Therefore, the “competency questions” may more accurately be termed “competency instructions”.

Competency questions about the structure of the tree:

1. Construct a simplified fact-tree showing only the dependencies between explananda.

2. Construct a simplified tree containing just the hierarchy of knowledge bases where the facts are derived from.
3. Construct a simplified tree showing only the explanations that extract their explanans from a machine learning model or Bayesian network. Explanations that contain no explanans produced by an external ML model or BN are derived by Plasido's rule-based reasoner. This competency questions removes the nodes that are produced solely by Plasido. Only the explanations that are conceptualised by at least one external KB are retained.

In order to be able to construct the answers to these queries, we need to be able to query the tree on a precise level. So, an ontology that constructs an arbitrary tree structure should form the basis of the explanation tree. This structure should be able to answer arbitrary questions about the depth, size, width, and similar properties of the tree.

4.4.3 Knowledge bases

The knowledge base ontology module is simpler than the hierarchy module and the fact module previously described. We merely ask to which category of KBs an instance belongs instead of asking its specific structure. The KB may exhibit a structure, but the ontology is not concerned with the structure of the KB, because the explanation dissolves the need to inspect the KB. Therefore, we only state two competency questions. The first asks which general type of KB the instance belongs to. The second requests other more specific types that identify the instance.

1. Is the KB a machine learning model, a Bayesian network, or an expert system?
2. Which combination of KB subcategories is the instance a type of?

Summary

In this chapter we ended the first phase of SABiO by stating our purpose and goals, the requirements, the ontology modularisation, and the competency questions. The former two look back on the previous chapters. The purpose and goals were a restatement of the introduction. The requirements were a summary of the main points extracted from the domain survey. The latter two provide a guide forward. The ontology modularisation gives a preliminary structure of the ontology. The competency questions indicate when the ontology can be considered sufficient. Namely, when the competency questions formalised in a query language can extract the requested knowledge from the ontology instance.

Chapter 5

The technique: Description Logic

In this chapter we will describe Description Logic (DL), the formalisms underlying conceptualisations. We will only cover the theory as far as relevant for our purposes. So, we will not define the formal semantics, but we will explain it in natural language terms. The first section introduces the standard Description Logic \mathcal{ALC} . The second section will introduce additional constructs that increase the expressiveness of \mathcal{ALC} .

5.1 Syntax and semantics

Description Logic provides a formal language to describe subsumption relations between concepts. Description Logic \mathcal{ALC} (Attributive (Concept) Language with Complements) is considered the basic instance, but can even be further reduced to \mathcal{AL} , the Attributive (Concept) Language without negation. It talks of concepts C , roles r and individuals a . The concepts define all groups and subgroups of entities that exist in a domain. The roles define the relations and functions that exist among concepts and individuals. The individuals are instances belonging to certain groups, and having particular roles.

For example, in the domain of knowledge base explanation we might want to state that each explanation has an explanandum and an explanans. Our model then consists of at least three concepts (Explanation, Explanandum and Explanans) and two roles (*hasExplanandum* and *hasExplanans*). An Explanation instance would require three individuals: an Explanation individual, an Explanandum individual, and an Explanans individual. The Explanation has its *hasExplanandum* role filled by the Explanandum individual, and its *hasExplanans* role filled by the Explanans individual. In the rest of this section we will show how we define in DL it necessary for an explanation to have both an explanandum and an explanans.

We base our further survey on Baader et al. (2017). Our definition 1 is a direct citation of them, because they provide a comprehensive textbook on Description Logic. We will give a survey of the basic logic \mathcal{ALC} , before extending it to increase its expressive power. To construct a concept description we use the following definition (Baader et al. 2017, Definition 2.1).

Definition 1. *Let C be a set of concept names and R be a set of role names disjoint from C . The set of \mathcal{ALC} concept descriptions over C and R is inductively defined as follows:*

- Every concept name is an \mathcal{ALC} concept description.
- \top and \perp are \mathcal{ALC} concept descriptions.
- If C and D are \mathcal{ALC} concept descriptions and r is a role name, then the following are also \mathcal{ALC} concept descriptions:

$C \sqcap D$ (conjunction),
 $C \sqcup D$ (disjunction),
 $\neg C$ (negation),
 $\exists r.C$ (existential restriction), and
 $\forall r.C$ (value restriction).

With a definition of an \mathcal{ALC} concept, we can construct an \mathcal{ALC} general concept inclusions (GCI) to define necessary ($C \sqsubseteq D$) and sufficient conditions ($D \sqsubseteq C$) for a concept C . This also allows us to define the necessary *and* sufficient conditions with the shorthand $D \equiv C$ for $C \sqsubseteq D \wedge D \sqsubseteq C$.

This way, we can formalise the requirement that each Explanation necessarily consists of both an Explanandum and an Explanans:

$$\text{Explanation} \sqsubseteq \exists \text{hasExplanandum.Explanandum} \sqcap \exists \text{hasExplanans.Explanans}$$

In other words, an Explanation is anything that resides in the intersection of the things that have an Explanandum as *hasExplanandum*-filler and have an Explanans as *hasExplanans*-filler. In a set theoretic description, an explanation is something that is a member of both the set of objects that have an Explanans and the set of objects that have an Explanandum.

We can also use the disjunction: $\text{Explanation} \sqsubseteq \text{Concept1} \sqcup \text{Concept2}$. This means the concept Explanation necessarily belongs to Concept1 or Concept2. It is important to note that Concept1 and Concept2 are assumed to be disjoint. If we want Concept1 and Concept2 to be disjoint, we have to explicitly specify that.

The existential restriction in

$$\exists \text{hasExplanandum.Explanandum}$$

is meant to denote the concepts that have at least one Explanandum that fills a *hasExplanandum*-role. The universal restriction

$$\forall \text{hasExplanandum.Explanandum}$$

denotes the concepts for which all *hasExplanandum*-roles are filled by an Explanandum. It does not exclude concepts with other roles than *hasExplanandum*. It merely specifies that any role of the type *hasExplanandum* should be filled by an Explanandum. So, as a special case, an element without roles satisfies any universal clause.

5.2 Additional expressive power

The GCI only allows the definition of subset relations, so we need to extend its expressive capabilities to include, among other, transitive relations. All extensions we describe

below strictly increase the expressivity of \mathcal{ALC} . We will not cover the formal proofs for each extension, these can be found in Baader et al. (2017, p. 37). Several more extensions exist, but we will use these during formalisation of the ontology.

Transitive roles: The \mathcal{ALC} logic cannot define a *hasDescendant* relation that is valid for any two elements related via an arbitrary long chain of *hasChild* roles. We therefore need to define *hasDescendant* as a transitive role:

$$\text{Trans}(\text{hasDescendant})$$

The idea of the proof is to build a bijection between two \mathcal{ALC} models. This shows that both can be considered functionally equivalent. Then we show for some sentence containing a transitive role that it is false on one model and true on the other model in spite of the bijection. This result leads us to conclude that the addition of transitive roles to \mathcal{ALC} strictly increases its expressiveness.

SELF: There exists no way to define a concept that necessarily has some role filled by itself. This construct also allows the construction of recursive roles.

Inverse roles: We denote the inverse of property r with r^- , which means that if two concepts c_0 and c_1 exist with $\langle c_0 \text{ r } c_1 \rangle$, then also $\langle c_1 \text{ r}^- c_0 \rangle$. We can also manually define *isExplanansOf* to be the inverse of *hasExplanandum*. This addition allows us to define symmetric roles. It also allows us to define concepts that are related via the inverse of an existing relation.

(Qualified) number restrictions: Number restrictions allow us to specify the exact number of fillers of a specific role. For example, we can define that each Explanation has at most one filler through an *hasExplanandum* role:

$$\text{Explanation} \sqsubseteq \leq 1 \text{ hasExplanandum}$$

When the restriction is qualified it allows us to specify the concept for which the number restriction counts. We can then specify that an Explanation has at most one *hasExplanandum*-filler that is an Explanandum concept:

$$\text{Explanation} \sqsubseteq \leq 1 \text{ hasExplanandum.Explanandum}$$

(Complex) role inclusion: This means that a tuple of two roles is subsumed by another role. For example, *isParent* combined with *isBrother* is subsumed by *isUncle*. In the explanation domain, we might state that an Explanandum is related to an Explanans via the Explanation. We would write both examples as follows:

$$\text{isParent} \circ \text{isBrother} \sqsubseteq \text{isUncle}$$

$$\text{hasExplanans}^- \circ \text{hasExplanandum} \sqsubseteq \text{performsExplanationOf}$$

Summary

In this chapter we gave the formal syntax and the informal semantics of Description Logic, such that we can use it to construct the reference ontology. We also defined some additional concepts to increase the expressiveness of Description Logic. This allows us to add inverse roles and complex role inclusions, among others, to our Description Logic axiomatisation.

Chapter 6

Reference ontology design choices

In this chapter we will argue for our specific construction of the explanation ontology. It functions as the second phase of SABiO (figure 6.1), where a formal axiomatisation is constructed from natural language constraints. Therefore, we will describe the structure we envision in natural language, which we will simultaneously translate into Description Logic. The chapter contains a dedicated section for each of the ontology modules, starting with single fact explanation and ending with the explanation hierarchy.

Referring to established foundational ontologies can increase the quality of the ontology (Gangemi and Presutti 2009), since we then refer to concepts that already have been tested in practice. Furthermore, constructing ontologies that are based on Ontology Design Patterns (ODP) is considered a good design practice from the viewpoint of ontology construction (Gangemi and Presutti 2009). Therefore, we will begin this chapter with a survey of earlier attempts at constructing an explanation ontology, such that we can build upon earlier attempts when constructing our ontology in the second half of this chapter.

6.1 Existing explanation ontologies

We found just two papers on the intersection of explanation and ontologies: Su et al. (2003) and Tiddi et al. (2015). The paper by Su et al. observes the problem from an agent perspective, interpreting a federated knowledge environment as a multi-agent

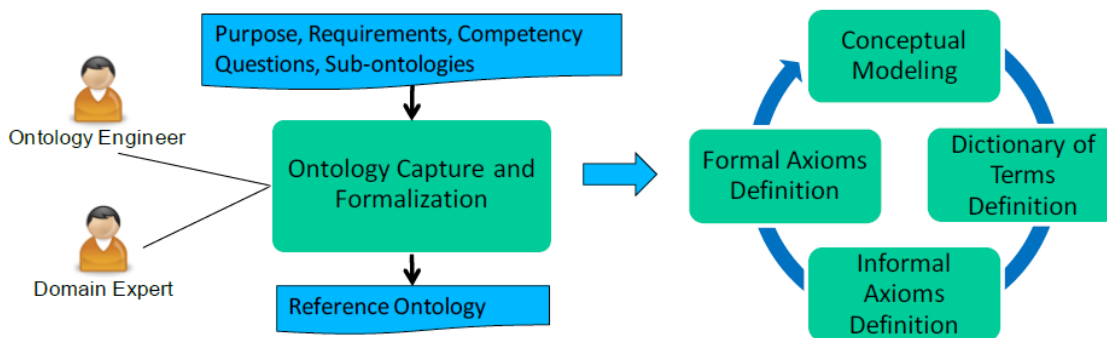


Figure 6.1: The second phase of SABiO (Falbo 2014)

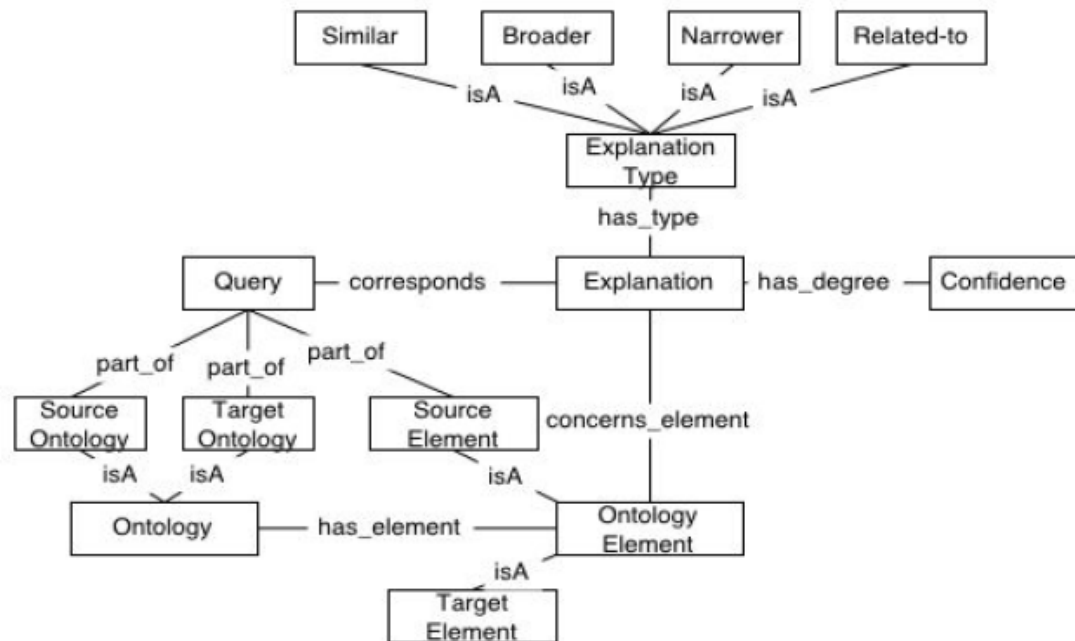


Figure 6.2: A model of the main concepts in Su et al.'s explanation profile

system. Their model is given in figure 6.2. An agent can give four types of explanations: 1) similar, 2) broader, 3) narrower, or 4) related-to. However, these explanations do not give an idea *why* something is the case. The goal of Su et al. is to link various independent ontologies adhering to different standards. While this may seem similar to our problem, the aim of our explanation ontology is not to show the similarities between knowledge bases. Our ontology needs to incorporate explanations for the output of a KB. Moreover, our KBs are explicitly allowed to be as diverse as is the case. It is only the output of a KB that has to adhere to the standard.

Neerincx et al. (2018), written by TNO-colleagues, also consider using Su et al. for their research in constructing a framework for human-agent team performance. They consider Su et al.'s ontology to be lacking in capturing goal based explanations, and consider the ontology to be unable to capture reception of explanation. They use similar arguments against Tididi et al. (2015). We do not consider that a drawback for our purposes, because our main goal is to model the contents of an explanation rather than the reception. Therefore, Neerincx et al.'s objections give us no reason against using Tididi et al.'s framework.

Indeed, Tididi et al.'s paper proved to be an important starting point for our research. Their ontology for explanation is shown in figure 6.3. They use it to produce an explanation in their linked data framework Dedalo¹. They found a definition from linguistics especially helpful:

When X happens, then, due to a given set of circumstances C, Y will occur because of a given law L.

¹The site previously hosting Dedalo is down and the hyperlink in Tididi et al. (2015) is broken. The OWL-file is available online: <http://ontologydesignpatterns.org/wiki/Submissions:ExplanationODP>.

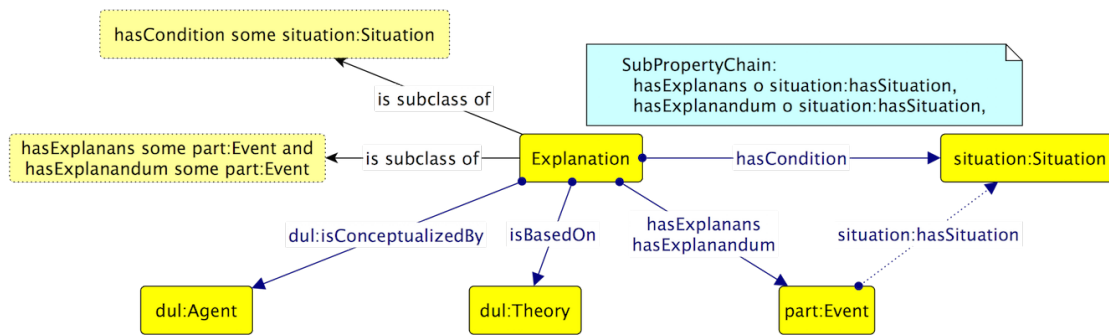


Figure 6.3: Tididi et al.'s ontology

We may wonder why event X happens at a particular time. In two steps the framework tries to construct the most likely explanation. First, it obtains a set of possible explanantia by gathering all statistically related events. Then, it decides on the best explanans by looking whether the event and the explanans appear in the same context. This procedure is formalised by computing the distance of the linked data path connecting both.

Also, the authors present their ontology as an Ontology Design Patteren (ODP), intending it to be re-used in other frameworks. In figure 6.3 it may seem that the concepts Situation, Part, Event, and Theory are left undefined. However, all four concepts are taken from the DOLCE + D&S Ultra Lite ontology (Presutti and Gangemi 2016). Therefore, we will use Tididi et al.'s ODP as a starting point for our ontology.

6.2 Fact explanation

We showed the design by Tididi et al. to be a good starting point. However, we can also pinpoint some limitations with Tididi et al.'s design.

Explanandum and Explanans as an Event: Tididi et al. have not given dedicated concept classes to the Explanandum and the Explanans, but have defined both as simply an Event, where the Event class is inherited from a foundational ontology. The explanation produced by the KBs in our use case is static, and therefore not suited as a subset of an Event concept.

A Situation as *hasCondition* filler: The second necessary condition for an Explanation is having a Situation, inherited from the foundational ontology, as a *hasCondition* filler. No concept in our domain plays the same role. If we would extend Tididi et al.'s ontology, we would need to give each Explanation a Situation as *hasCondition* filler.

Even though the general structure of Tididi et al.'s ontology fits our domain, their necessary conditions have no correspondence in our domain. So, we will not directly extend their ontology, but will draw inspiration from it. In particular, we will retain the Explanation as the upper concept that has both a *hasExplanandum* and a *hasExplanans* relation. However, we fill these roles with an Explanandum and an Explanans respec-

tively. We will also retain the role *isConceptualizedBy*² from the Dolce+D&S Ultralite foundational ontology (Presutti and Gangemi 2016), since we consider the explanation to be conceptualised by the particular KB. In Description Logic formalisation the basis will look as follows:

$$\begin{aligned} \textit{Explanation} \equiv \exists \textit{hasExplanandum.Explanandum} \sqcap \\ \exists \textit{hasExplanans.Explanans} \end{aligned} \quad (6.1a)$$

$$\textit{Explanation} \sqsubseteq \exists \textit{isConceptualizedBy.KnowledgeBase} \quad (6.1b)$$

6.2.1 Explanandum ontology

As we said in the previous section, Tiddi et al. have merely defined an Explanandum as an Event, which has too little internal structure for our purposes. Although we have stated reuse as a goal, it is not a goal in itself. We take from Tiddi et al.'s ontology only what fits our purposes.

From the literature two types of explananda emerged: 1) contrastive and 2) simple. The former consists of an observed fact and an expected foil. The explanation explains the Fact through a contrast between fact and foil. A simple explanandum is just a fact or rule that is explained by its corresponding explanans. These two forms of explananda are disjoint: a simple explanandum cannot also be a contrastive explanandum. We translate these statements into Description Logic:

$$\textit{Explanandum} \sqsubseteq \textit{SimpleExplanandum} \sqcup \textit{ContrastiveExplanandum} \quad (6.2a)$$

$$\textit{SimpleExplanandum} \sqsubseteq \neg \textit{ContrastiveExplanandum} \quad (6.2b)$$

$$\textit{SimpleExplanandum} \sqsubseteq \textit{Fact} \sqcup \textit{Rule} \quad (6.2c)$$

$$\textit{ContrastiveExplanandum} \sqsubseteq \exists \textit{hasFoil.Fact} \sqcap \exists \textit{hasFact.Fact} \quad (6.2d)$$

6.2.2 Explanans ontology

In our statement of the requirements we found that the ontology should capture the four types of model explanation: 1) explanation by explainable proxy, 2) outcome explanation, 3) model inspection, or 4) a transparent box. These four categories are the parts that explain something. So, the union of these classes necessarily is an explanans.

$$\begin{aligned} \textit{ProxyExplanation} \sqcup \\ \textit{OutcomeExplanation} \sqcup \\ \textit{ModelInspection} \sqcup \\ \textit{TransparentBox} \sqsubseteq \textit{Explanans} \end{aligned} \quad (6.3)$$

We explicitly model a sufficient condition for the Explanans concept instead of a necessary condition, the converse. First of all, we do not want to restrict the types of

²The role is taken from the Dolce+D&S Ultralite foundational ontology, where it is written in American English with a “z”. When do not mean the concrete role but use the word in a general sense, we will write the word with an “s” as is customary in British English.

explanantia to these four types. More types may be developed, or may be necessary for a future use case. Moreover, we want to allow an Explanans to belong to no sub-category if the type of explanans does not belong to any established type.

The sufficient condition ensures that any concept belonging to one of the four categories is necessarily an Explanans, which it can only be if it performs the explanation of some fact. In other words, the Explanans needs to be connected to an Explanation concept. In chapter 5 we defined the inverse relation, which we use to formalise that an Explanans should have an incoming *hasExplanans* role from an Explanation.

$$Explanans \sqsubseteq \exists hasExplanans^- .Explanation \quad (6.4)$$

In the literature survey we found three types of explainable proxies, or, recalling our discussion on algorithmic transparency (section 2.1.2), what some researchers suppose are explainable proxies: 1) linear model, 2) decision tree, and 3) decision rules. It is not relevant if these proxies are indeed as naturally interpretable as mainly Guidotti et al. (2018) argues, or if these proxies quickly lose their interpretability when increasing in complexity (Lipton 2016). We would like to remind the reader that our goal is to increase the explainability through interoperability instead of by improving one specific method.

We already used the ProxyExplanation concept in equation 6.3, but we have not yet defined what exactly constitutes a ProxyExplanation. In natural language we can state that something is a ProxyExplanation if it has an explainable proxy, of which we have defined three types. This gives rise to the following two equations. The second equation is a sufficient condition instead of a necessary condition since other types of explainable proxies may be developed.

$$ProxyExplanation \sqsubseteq \exists hasExplainableProxy .ExplainableProxy \quad (6.5a)$$

$$LinearModel \sqcup DecisionTree \sqcup DecisionRules \sqsubseteq ExplainableProxy \quad (6.5b)$$

For the subtype of outcome explanation the literature gave rise to four types. We will immediately write the four types into the DL formula instead of first enumerating them in natural language, since we consider the process to be clear by now.

$$\begin{aligned} FeatureImportance \sqcup \\ PrototypeSelection \sqcup \\ SalienceMap \sqcup \\ LocalExplanation \sqsubseteq OutcomeExplanation \end{aligned} \quad (6.6)$$

For the same reasons as with the GCI for Explanans and for ProxyExplanation, equation 6.6 is also modelled as a sufficient condition for OutcomeExplanation instead of a necessary condition.

The LocalExplanation concept is intended to capture any explanation where the classifier is applied locally: a set of rules from the rule-base that together allow the derivation of the explanandum, the path extracted from a decision tree that shows how

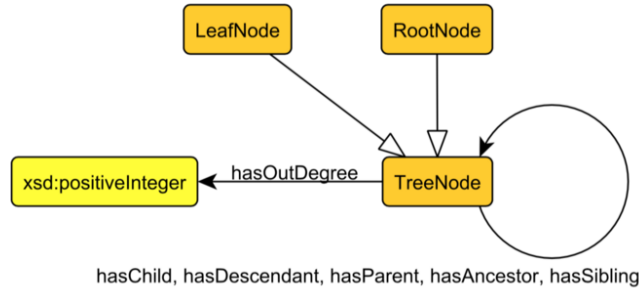


Figure 6.4: Tree ontology design pattern (Carral et al. 2017)

the tree was traversed to reach a particular conclusion, or a decision tree trained on a local sample.

$$\begin{aligned}
 & DecisionRuleDerivation \sqcup \\
 & DecisionTreePath \sqcup \\
 & DecisionTree \sqsubseteq LocalExplanation
 \end{aligned} \tag{6.7}$$

This leaves *ModelInspection* the only clause of equation 6.3 to be defined. We model it as a necessary condition instead of a sufficient condition as we did with most previous types, because any way of presentation can be captured in either images or text. So, the concept captures possible improvements in providing textual or graphical explanation.

$$ModelInspection \sqsubseteq TextualInspection \sqcup GraphicalInspection \tag{6.8}$$

It may strike the reader that the DL formulas of this section have left quite some concepts undefined. For example, in equation 6.6 we gave a sufficient condition for *OutcomeExplanation*, but we have not defined what either *FeatureImportance* or *PrototypeSelection* exactly is. In the following we will define in the DL formalism the remaining important relations and concepts. Very detailed but nevertheless intuitive concepts we will only define in the operational ontology.

6.2.3 Tree-based structures

We will first tackle the tree structure that is needed for tree-based proxies and for outcome explanation via a tree path. We found an Ontology Design Pattern that fits our purposes. Carral et al. (2017) produced an ontology for the abstract structure of a tree. This ODP does not make assumptions that run against our proposed usage. We need to specify which (combination of) relations form the edges of the tree, and which (combination of) entities form the nodes.

Figure 6.4 shows that we have access to the following concepts: 1) leaf node 2) root node, and 3) tree node. We can also employ the *hasChild* relation along with its family members 1) *hasDescendant*, 2) *hasParent*, 3) *hasAncestor*, and 4) *hasSibling*.

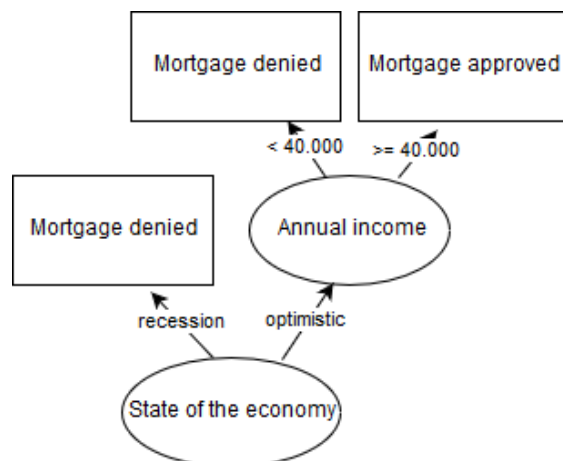


Figure 6.5: A small decision tree for deciding someone's mortgage

Figure 6.5 gives a small example of a decision tree. It shows that any node is labelled with the feature on which is being split. Also, each edge is labelled with a split value, which can be either numerical or categorical. Moreover, a decision tree should completely consist of decision tree nodes. A mix of *DecisionTreeNode*s with other types of nodes is an unwanted mix of concepts.

$$DecisionTreeNode \sqsubseteq TreeNode \sqcap \exists hasLabel.Feature \quad (6.9a)$$

$$DecisionTreeEdge \sqsubseteq hasChild \sqcap (\exists hasValue.FeatureClass \sqcup \exists hasValue.Numerical) \quad (6.9b)$$

$$DecisionTree \equiv DecisionTreeNode \sqcap \forall hasChild.DecisionTreeNode \quad (6.9c)$$

We will not put restrictions on the number of edges per node. The tree may be binary, or whatever suits the needs. In one case we may need to present the complete tree as a form of model explanation, or model inspection. In another case, we may simply give a path in the tree as outcome explanation. This would be a unary subtree from a > 1 -ary decision tree.

$$DecisionTreePath \sqsubseteq DecisionTree \sqcap < 2 hasChild.DecisionTreeNode \sqcap \forall hasChild.DecisionTreePath \quad (6.10)$$

6.2.4 Rule-based structures

We can also use the tree structure ODP for a rule-based explanation. We see a rule derivation ($p \rightarrow q, p/q$) as a tree with the conclusion or consequent q as the root, the rule $p \rightarrow q$ as a child and the antecedent p (or antecedents) as the remaining children. Both the antecedent and the rule may themselves require an explanation.

The structure of a rule-based explanation is different in a model explanation context than in an outcome explanation context. In the former, a set of rules is presented as a simplified proxy. In the latter, a rule derivation is presented with the explanandum

as the root. On the one hand we have a set rules, while on the other hand we have a derivation tree of an arbitrary depth and size. We first model a simple set of rules.

$$DecisionRules \sqsubseteq \exists hasRule.Rule \sqcap \forall hasRule.Rule \quad (6.11)$$

To define a rule derivation we need to use the tree ODP, because a rule derivation has a hierarchical structure instead of being a bag of items like DecisionRules. A derivation necessarily has a fact child and a rule child.

$$DerivationNonLeafNode \sqsubseteq TreeNode \sqcap \exists hasChild.FactNode \sqcap \\ \exists hasChild.RuleNode \sqcap \forall hasChild.DerivationNode \quad (6.12a)$$

$$DerivationLeafNode \sqsubseteq FactNode \sqcap LeafNode \quad (6.12b)$$

6.2.5 Feature importance, salience maps and prototype selection

In this section we will describe the structure of the local explanation algorithms Feature Importance, Salience Map and Prototype Selection, because the three algorithms share remarkable similarities. Instead of a single local explanation in the form of a tree path which uses the structure of the classifier, a set of explanation items is handed to the user. Feature Importance gives the most decisive features, Salience Mapping shows which properties of the data elicit a strong response, and Prototype Selection returns the most similar data set entry. Moreover, mixing Prototype Selection and Feature Importance into one explanation algorithm has been shown to be feasible by Adhikari (2018).

As we showed in figure 2.2, an item of a feature importance explanation is characterised by three variables: 1) the name of the feature class, 2) the exact value of the decisive feature, and 3) a measure of the strength of the influence (i.e. how decisive was the feature). A Salience Map item uses some of the same roles (figure 2.3): 1) the name of the feature class, 2) the exact value of the feature, and 3) a measure of the certainty of the classification. Concluding, a Prototype Selection item also uses some of the same roles: 1) the prototype from the training set, 2) an integer showing the classification label of the prototype, and 3) a measure of the strength of the similarity.

This way, we can reuse some roles for all three concepts. Equations 6.13a through 6.13c define the types of explanations consisting of a set of items. Equation 6.14a defines a measure indicating the strength of the association, which is shared by all three types. Equation 6.14b defines the roles unique for a PrototypeSelectionItem.

$$FeatureImportanceExplanation \sqsubseteq \\ \exists hasFeatureImportanceItem.FeatureImportanceItem \quad (6.13a)$$

$$PrototypeSelectionExplanation \sqsubseteq \\ \exists hasPrototypeSelectionItem.PrototypeSelectionItem \quad (6.13b)$$

$$SalienceMapExplanation \sqsubseteq \\ \exists hasSalienceMapItem.SalienceMapItem \quad (6.13c)$$

$$\begin{aligned} \text{FeatureImportanceItem} \sqcup \text{PrototypeSelectionItem} \sqcup \text{SaliencyMapItem} \\ \sqsubseteq \exists \text{hasStrength.Integer} \end{aligned} \quad (6.14a)$$

$$\begin{aligned} \text{PrototypeSelectionItem} \sqcup \text{hasPrototypeID.Integer} \\ \sqsubseteq \text{hasSupportingClassification.Integer} \end{aligned} \quad (6.14b)$$

6.3 Knowledge base ontology

The second ontology module we formalise in DL is the knowledge base module. In the literature we observed machine learning models, Bayesian networks, and rule-based expert systems. Other types of KBs exist and may be incorporated in the ontology at a later stage. Therefore, we model the concept inclusion as a sufficient condition.

$$\begin{aligned} \text{MachineLearningModel} \sqcup \text{ExpertSystem} \sqcup \text{BayesianNetwork} \\ \sqsubseteq \text{KnowledgeBase} \end{aligned} \quad (6.15)$$

The three knowledge bases are disjoint. A KB cannot be an instance of more than one of these subtypes.

$$\text{MachineLearningModel} \sqsubseteq \neg \text{ExpertSystem} \quad (6.16a)$$

$$\text{MachineLearningModel} \sqsubseteq \neg \text{BayesianNetwork} \quad (6.16b)$$

$$\text{ExpertSystem} \sqsubseteq \neg \text{BayesianNetwork} \quad (6.16c)$$

Especially in machine learning models quite some different types exist. In the papers we surveyed in chapter 2, next to a taxonomy for explanation, also a list of machine learning algorithms was presented. Guidotti et al. surveyed papers on explanation for the following KBs: 1) Neural network, 2) Deep neural network, 3) Tree ensemble, and 4) Support vector machine. They considered neural networks to be independent from *deep* neural networks, but we will consider the latter a subtype.

$$\begin{aligned} \text{MachineLearningModel} \sqsubseteq \text{NeuralNetwork} \sqcup \\ \text{TreeEnsemble} \sqcup \\ \text{SupportVectorMachine} \end{aligned} \quad (6.17a)$$

$$\text{DeepNeuralNetwork} \sqsubseteq \text{NeuralNetwork} \quad (6.17b)$$

The KBs Guidotti et al. considered inherently interpretable can also produce an explanation through one of the model-agnostic explainers: 1) Decision tree, 2) Rules, and 3) Linear model. So, the explainable proxies that are part of a ProxyExplanans, are themselves also explainable knowledge base instances.

$$\text{ExplainableProxy} \sqsubseteq \text{KnowledgeBase} \quad (6.18)$$

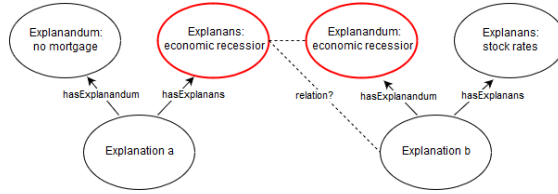


Figure 6.6: The explanandum of the subexplanation is identical the explanans of the superexplanation.

For ML the type of input is also important. It is necessary that the ML algorithm belongs to one of the following categories: 1) Supervised, 2) Semi-supervised, 3) Un-supervised, and 4) Reinforcement. We consider the machine learning domain to be encapsulated by these types, so we construct an equivalence instead of a concept inclusion. This does not mean a machine learning model that belongs to none of these categories is forbidden to exist. By the open-world assumption we can infer that the type of input is unknown instead of non-existent.

$$\begin{aligned}
 & \textit{Supervised} \sqsubseteq \\
 & \textit{SemiSupervised} \sqsubseteq \\
 & \textit{Unsupervised} \sqsubseteq \\
 & \textit{Reinforcement} \equiv \textit{MachineLearningModel}
 \end{aligned} \tag{6.19}$$

This taxonomy can be made even more detailed, but a more detailed taxonomy is not necessary for our purposes. We should not include arbitrary ML algorithms that do not occur in the relevant literature. The open world assumption allows additional concepts to extend the KnowledgeBase concept.

6.4 Explanation tree ontology

The central part of the ontology is the construction of the explanation hierarchy. We again use the ODP for tree structures (figure 6.4) from Carral et al. (2017) as our basis. The ODP leaves us to define what exactly constitutes an edge and what constitutes a node.

Let us consider two explanations where one is directly dependent on the other (figure 6.6). The explanans of the upper explanation may itself require an explanation. However, if the explanans requires an explanation, it becomes the explanandum of yet another explanation. This way, a fact can fulfil the role of both explainer as well as thing being explained. The “economic recession” node of the example fulfils the role of both explanans and explanandum. It explains why someone’s mortgage application is rejected, and is itself explained by the stock rates.

Our goal is to link these singular explanations together. Figure 6.7 gives a naive implementation. We explicitly relate an Explanans part to the Explanation where it has an identical Explanandum via the *hasIdenticalExplanandum* relation. This way, the Explanans of the first Explanation and the Explanandum of the second Explanation

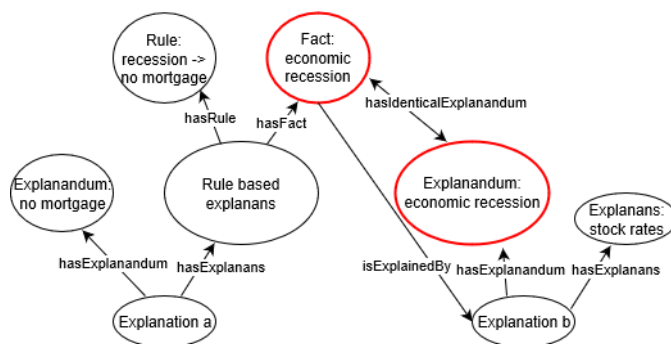


Figure 6.7: Modelling problem. An identical entity occurs twice.

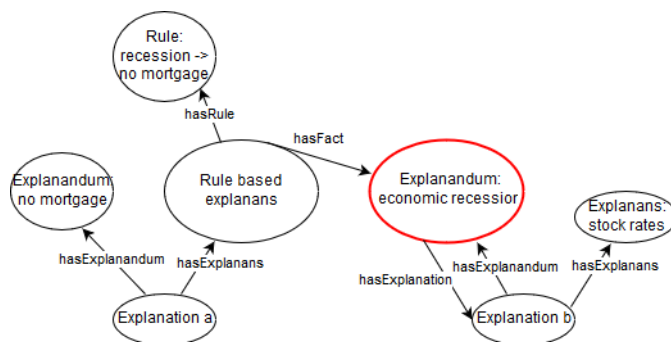


Figure 6.8: Modelling solution. We collapse the identical entities to show that $hasExplanans \equiv hasExplanation^-$.

may have different unique identifiers, but are inferred to be equal via the *hasIdenticalExplanandum* role. However, referring to both with the same unique identifier would be better, because both are in fact identical.

The improved modelling solution is given in figure 6.8. We collapse the Fact and the Explanandum into one concept. This relieves us of the unnecessary and superfluous *hasIdenticalExplanandum* relation. In technical terms we introduce in the next chapter, we will give both the same Universal Resource Identifier (URI). With this solution, the structure becomes less complicated. The function of the *hasExplanation* role also becomes clear as the inverse of *hasExplanandum*.

$$hasExplanation \equiv hasExplanandum^- \quad (6.20)$$

Having defined how we will connect several independent Explanations, we can define an Explanation to function as the node of an explanation tree. After all, the Explanation concept is the main source that is connected to all parts of the explanation.

$$ExplanationTreeNode \sqsubseteq Explanation \sqcap Node \quad (6.21)$$

Since an Explanation necessarily has an Explanandum and Explanans, by transitivity of the subset relation the ExplanationTreeNode also has an Explanandum and Explanans. This leaves us to define the edge linking one Explanation to its child concepts.

6.4.1 Descendant relations

The variety of Explanans subconcepts complicates the linking of explanations. In figure 6.8 we can observe that a path between two Explanation object follows three roles: 1) *hasExplanans*, 2) *hasFact*, and 3) *hasExplanation*. We can also write the third role as an inverse role: 1) *hasExplanans*, 2) *hasFact*, and 3) *hasExplanandum⁻*. The Explanation class necessarily has both a *hasExplanans* filler and a *hasExplanandum* filler. Only the *hasFact* role is specific to a type of Explanans. So, we need to make a generalised role for *hasFact*, *hasRule*, and other parts of the Explanans that may themselves need an explanation. We call the general role *hasExplainablePart*, to indicate that the Explanans consists of several parts that themselves may need to be explained.

$$hasFact \sqcup hasRule \sqsubseteq hasExplainablePart \quad (6.22)$$

All explanans parts for other explanantia should also be subconcepts of the role *hasExplainablePart*:

$$\begin{aligned} &hasPrototypeSelectionItem \sqcup \\ &hasFeatureImportanceItem \sqcup \\ &hasSalienceMapItem \sqcup \\ &hasExplainableProxy \sqcup \\ &hasLocalExplanation \sqsubseteq hasExplainablePart \end{aligned} \quad (6.23)$$

This allows us to generalise the three roles that link an Explanation to its children.

$$\begin{aligned} &ExplanationTreeEdge \sqsubseteq \\ &hasExplanans \circ hasExplanansPart \circ hasExplanandum^{-} \end{aligned} \quad (6.24)$$

It leaves us to define what a Leaf of the explanation tree is. In a conversational explanation we can ask “why?” *ad infinitum*, but a computationally produced explanation should start with a set of axioms. If the explanation procedure is well defined, these starting points should be self-explanatory. We will take this literally and model an Axiom to have itself as its Explanans.

$$Axiom \sqsubseteq \exists hasExplanans.SELF \sqcap Explanation \quad (6.25a)$$

$$ExplanationTreeLeaf \sqsubseteq Axiom \quad (6.25b)$$

6.4.2 Revisiting rule-based explanation

Figure 6.9 shows the modelling solution we provided for rule derivation before we constructed the ExplanationTree concept. We want to revisit this modelling solution to reuse the ExplanationTree concept, because each derivation contained in a DerivationTree can also be seen as an explanation. The conclusion of the Derivation is the explanandum, while the rule and the facts make up the explanans. In figure 6.10 we show the modelling option were the explanation tree is reused.

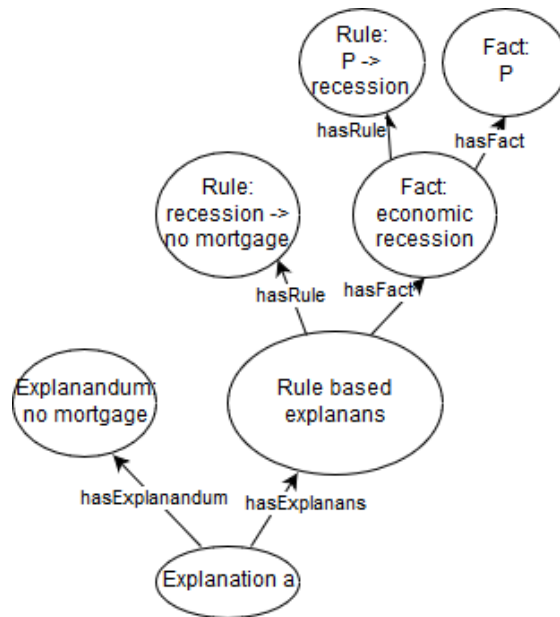


Figure 6.9: Initial modelling for rule-based outcome explanation

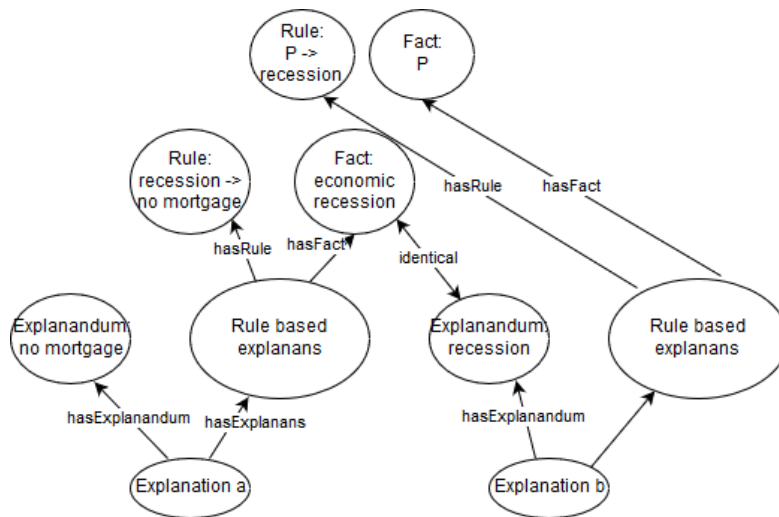


Figure 6.10: Optimal modelling through reuse of existing structures

The modelling solution with `ExplanationTree` reuse is an improvement, because it ensures that the same explanation structures is used throughout the tree. The previous modelling solution (figure 6.9) may be more readable and simpler. However, human readability is not a good argument for ontology construction, because the human readability is provided by the formalised competency questions. The reuse of structures eases the task of formalising competency question. Therefore, we model a derivation tree to be a type of explanation tree.

$$DerivationNode \sqsubseteq ExplanationNode \tag{6.26}$$

Summary

In this chapter we used Description Logic to formalise the important concepts of the KB explanation domain. An `Explanation` necessarily has an `Explanans` as *hasExplanans*-filler and an `Explanandum` as *hasExplanandum*-filler. It also has a `KnowledgeBase` as *isConceptualizedBy*-filler. The hierarchical structure of the explanation will be represented as an instance of the ontology design pattern defining a tree structure (Carral et al. 2017). The edges of the `ExplanationTree` are defined as a complex role inclusion of a *hasExplanans*, *hasFact*, and an inverse *hasExplanandum*, where a `Fact` plays the role of both explanandum of an explanation and part of the explanans for another explanation. In the next chapter we will turn these formulas into the computer-readable format of an ontology serialisation. We also check whether the ontology has the desirable properties of being decidable and complete.

Chapter 7

Operational ontology and implementation

In this chapter, we will turn the reference ontology into an operational ontology. The reference ontology is intended to be as precise to the domain as possible. In the operational ontology we will trade a decrease in precision for the useful computational properties of decidability and completeness. Our ontology is defined in the Web Ontology Language (OWL), which is the W3C standard for ontologies.

We will first introduce the reader to the concrete syntax or serialisation of an OWL-file. Then we will compare three OWL profiles that restrict the expressiveness to obtain good computational properties. We will explain that OWL DL fits our purposes best, because OWL Full places no restrictions, while OWL Lite places too many restrictions. This chapter will conclude with a graphical presentation of the several ontology modules we constructed that together form the Explanation ontology. We also created a small ontology for the mortgage domain, such that we can more easily visualise an ExplanationTree instance. The mortgage domain ontology will also be employed in the proof of concept of chapter 9. We conclude the chapter with an instance of the ExplanationTree.

7.1 Serialisation

The serialisation or concrete internal representation of an ontology is a set of triples:

```
?subject ?predicate ?object
```

The ?<name>-syntax denotes that the node is a variable. We can instantiate the variables to state that some entity called “Explanation” has another entity called “Explanans” as its *hasExplanans*-filler.

```
https://www.tno.nl/data/knowledgeBaseExplanation#Explanation
  https://www.tno.nl/ontology/knowledgeBaseExplanation#hasExplanans
  https://www.tno.nl/data/knowledgeBaseExplanation#Explanans
```

Each variable is uniquely identified with a Uniform Resource Identifier¹ (URI), which is a superclass of the more commonly known Uniform Resource Locator² (URL). The URI

¹<https://tools.ietf.org/html/rfc3986>

²<https://www.w3.org/Addressing/URL/url-spec.txt>

```

# Domain specific
@prefix tno: <https://www.tno.nl/ontology/knowledgeBaseExplanation#> .
@prefix tnoData: <https://www.tno.nl/data/knowledgeBaseExplanation#> .
@prefix tnoPoc: <https://www.tno.nl/ontology/knowledgeBaseExplanationPOC#> .
@prefix tnoPocData: <https://www.tno.nl/data/knowledgeBaseExplanationPOC#> .

# Ontology design patterns or foundational ontologies
@prefix dul: <http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#> .
@prefix tree: <http://www.odp.org/tree#> .

# Ontology semantics
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix terms: <http://purl.org/dc/terms/> .
    
```

Listing 7.1: We use these namespace prefixes used throughout the remainder of this thesis. The namespaces `tnoData` and `tnoPocData` are intended for individuals, whereas namespaces `tno` and `tnoPoc` are intended for other resources. The namespaces containing the infix `Poc` are for ontologies used in our proof of concept but not intended for a general explainability purpose.

can be divided into a namespace before the number sign (`#`) and a local name following it. The namespace functions as an identifier for the ontology as a whole. The local name specifies a resource contained in the ontology. Repeating the namespace for each triple gives us tediously long triples, so we commonly abbreviate the namespaces. Listing 7.1 gives the namespace prefixes we will use throughout the remaining chapters. This allows us to shorten the above triple to

```
tnoData:Explanation tno:hasExplanans tnoData:Explanans
```

Each ontology resource can perform one of the following roles.

Concept: A concept is a subset of the domain that adheres to a particular set of restrictions. An instance of a concept is called an individual.

DataType: A datatype is a set of values, for example integers, that exist independently of the domain. An instance of a datatype is called a literal.

Object property: A relation between two concepts is called an object property.

Datatype property: A relation between a concept and a datatype is called a datatype property.

Individual: An instance in the domain is called an individual.

The ontologies the prefixes (figure 7.1) refer to are categorised as follows.

Domain specific ontologies: In our case we have the Explanation ontology, of which we already described the DL formalisation in chapter 6. Prefixes `tno` and `tnoData` refer to this ontology. In this chapter we will introduce an ontology of the Mortgage domain to be used in our proof of concept. This ontology is referred to by namespaces `tnoPoc` and `tnoPocData`.

Foundational ontologies and ontology design patterns: A design principle of ontology construction is to reuse existing design where possible (Gangemi and Presutti 2009). The prefix `dul` refers to the Dolce + D&S Ultralite foundational ontology (Presutti and Gangemi 2016), which contains implemented concepts for general entities. The prefix `tree` refers to the design pattern for tree based structures (Carral et al. 2017).

OWL and RDF specifications: The remaining namespaces provide the ground concepts an ontology is made up of. For example, the *subClassOf*-relation and the *equivalentClass*-relation have their own respective unique URIs:

```
rdfs:subClassOf
owl:equivalentClass
```

The different prefixes show that both relations are taken from a different namespace. The OWL namespace contains more intricate relations than the RDFS namespace.

7.2 OWL profiles

The constructs provided by OWL and the Resource Description Framework Schema (RDFS) allow the construction of a model that is decidable nor complete. In our case this is undesired. There may exist an input for which the reasoner does not terminate (undecidable), and there may exist formula ϕ for which neither ϕ itself nor its complement $\neg\phi$ can be derived (incompleteness). Three OWL-profiles (OWL-species in outdated terminology) constrain the use of too expressive constructs.

OWL Lite: McGuinness and Harmelen (2004) describe the complete set of OWL Lite restrictions. The profile is sufficient for ontologies that need little in way of reasoning capacities, such as our Mortgage ontology we describe later in this chapter. The object properties OWL Lite forbids the use of are, among others, *owl:unionOf* and *owl:disjointWith*. OWL Lite also restricts cardinality constraints to 1 or 0. We used the *owl:unionOf* role in equation 6.16, so our formalisation does not follow the OWL Lite constraints.

OWL DL: This profile puts the least of restrictions on OWL without making the ontology possibly undecidable or incomplete. First of all, it requires a separation between types. In other words, a unique name cannot be shared by a class and an individual or property. The ontology also has to adhere to the global restrictions stated in Motik et al. (2012, sec. 11), which we will explain shortly.

OWL Full: The OWL Full profile puts no restrictions on the ontology. However, we can no longer use a reasoner to infer additional relations from our asserted relations, because the ontology is not guaranteed to be decidable and complete. We want to use a reasoner to infer subclass relations, inverse properties and possible equivalent classes. OWL Full does therefore not fit our needs.

It is not uncommon that the ontology strictly following the Description Logic formalisation is not in OWL DL. Carral et al. (2017) also had to weaken their DL formalisation

of a tree structure to keep the ontology decidable. The following DL formulas are part of their axiomatisation:

$$\textit{hasChild} \sqsubseteq \textit{hasDescendant} \quad (7.1a)$$

$$\textit{Irreflexive}(\textit{hasChild}) \quad (7.1b)$$

$$\textit{Irreflexive}(\textit{hasDescendant}) \quad (7.1c)$$

$$\textit{Trans}(\textit{hasDescendant}) \quad (7.1d)$$

OWL DL defines an object property to be *simple* if it has no direct or indirect subproperty that is transitive or defined as a property chain (Motik et al. 2012). OWL DL also defines a set of expressions, including irreflexivity, that are not allowed to contain a non-simple property. The DL axiomatisation (equation 7.1) does not follow this rule, because the irreflexive object property *hasDescendant* also is non-simple due to its transitivity. The TreeODP is brought in OWL DL by removing the irreflexivity of *hasDescendant* and *hasAncestor*.

The same restriction is the reason that our axiomatisation (chapter 6) is not in OWL DL. We used the Tree ODP to define the edges and nodes of an explanation tree. We recall the axiomatisation:

$$\textit{hasExplanans} \circ \textit{hasExplainablePart} \circ \textit{hasExplanation} \sqsubseteq \textit{hasExplanationChild} \quad (7.2a)$$

$$\textit{hasExplanationChild} \sqsubseteq \textit{hasChild} \quad (7.2b)$$

The *hasExplanationChild* object property is not a simple property, because it is defined as a chain of properties. The *hasChild* object property contains *hasExplanationChild* as a subproperty, so it is also a non-simple object property. In the Tree ODP it is defined to be irreflexible. Therefore, we have an object property that is both non-simple and irreflexible, which violates the OWL DL restriction.

The practical consequence for our ontology is that the *hasExplanationChild* role cannot be inferred from the three individual roles, but has to be asserted. Otherwise, it would be sufficient to define the *hasExplanans*, *hasExplainablePart*, and *hasExplanandum* roles, and let the *hasExplanationChild* relation be inferred. Since the property chain violates the OWL DL restriction, the construction procedure of the tree has to explicitly assert the edges of the tree.

7.3 Implemented ontology

We begin this section with a presentation in both text and graphics of the Explanation ontology we constructed from the Description Logic representation. The concrete OWL-file can be found at the TNO ontology repository³. It is too large to be explained line after line, so we will explain it through a series of graphical proxies. We will then show

³Repository: <http://ontology.tno.nl>
Our ontology: <http://ontology.tno.nl/KnowledgeBaseExplanation.owl>







Primitive	Application	Primitive	Application
	classes		datatypes, property labels
	properties		special classes/properties
	property directions		labels, cardinalities

Table 7.1: Meaning of symbols used in VOWL visualisations (Lohmann et al. 2016). Each class is coloured blue, with external classes a deeper shade of blue. Datatypes are coloured yellow. Datatype properties are coloured green.

the relatively simple Mortgage domain ontology. We implemented both ontologies using the Protégé editor (Musen and Protégé Team 2015). To conclude this section we show an Explanation individual for a rejected Mortgage application. In other words, we construct an ExplanationTree individual with an Explanation containing the following triple as its Explanandum:

```
tnoPocData:Corrie tnoPoc:isApproved "false"^^xsd:boolean
```

7.3.1 Explanation ontology

The graphical representation of the ontology was created with VOWL (Lohmann et al. 2016), except figure 7.2, which was constructed using the OntoGraf⁴ Protégé plugin. The meaning of each symbol is given in table 7.1. All figures present a hierarchy of classes instead of single individuals. Figure 7.1 gives the top level view containing the Explanandum with its subclasses, and the general concepts Explanation, Explanans, and Agent. These will be detailed in additional figures.

Figure 7.1 shows that the Explanandum has three subclasses, following the Description Logic formalisation. Additionally, the figure shows how we implemented a Rule and a Fact by giving the datatype properties. Since Plasido internally operates on triples, the fact we want to explain necessarily is in the form of a triple. So, the fact has to define a triple by containing a subject, predicate, and object.

In figure 7.2 we give a detailed view of the Agent class. The subclasses are not necessarily disjoint, since membership of some subclasses concerns the type of input data, while membership of other subclasses concerns the type of classification algorithm. We defined a Linear Model to be a subclass of both an Explainable Proxy and an Explainable ML model, so those two classes are necessarily not disjoint. Otherwise, a Linear Model individual cannot exist.

The Explanans class we explain in figures 7.3 and 7.4, with the former showing the general layout of all subclasses except OutcomeExplanans. The latter figure is solely dedicated to the OutcomeExplanans class and its subclasses. Figure 7.3 shows all subclasses, of which Axiom is disjoint from its siblings. The ExplainableProxy subclass shows that we modelled the three types of explainable proxies we considered in chapter 6. We reuse our existing concepts DecisionTree, LinearModel, and DecisionRules from the KnowledgeBase module, since a proxy is also a knowledge base. The DecisionTree class

⁴<https://protegewiki.stanford.edu/wiki/OntoGraf>

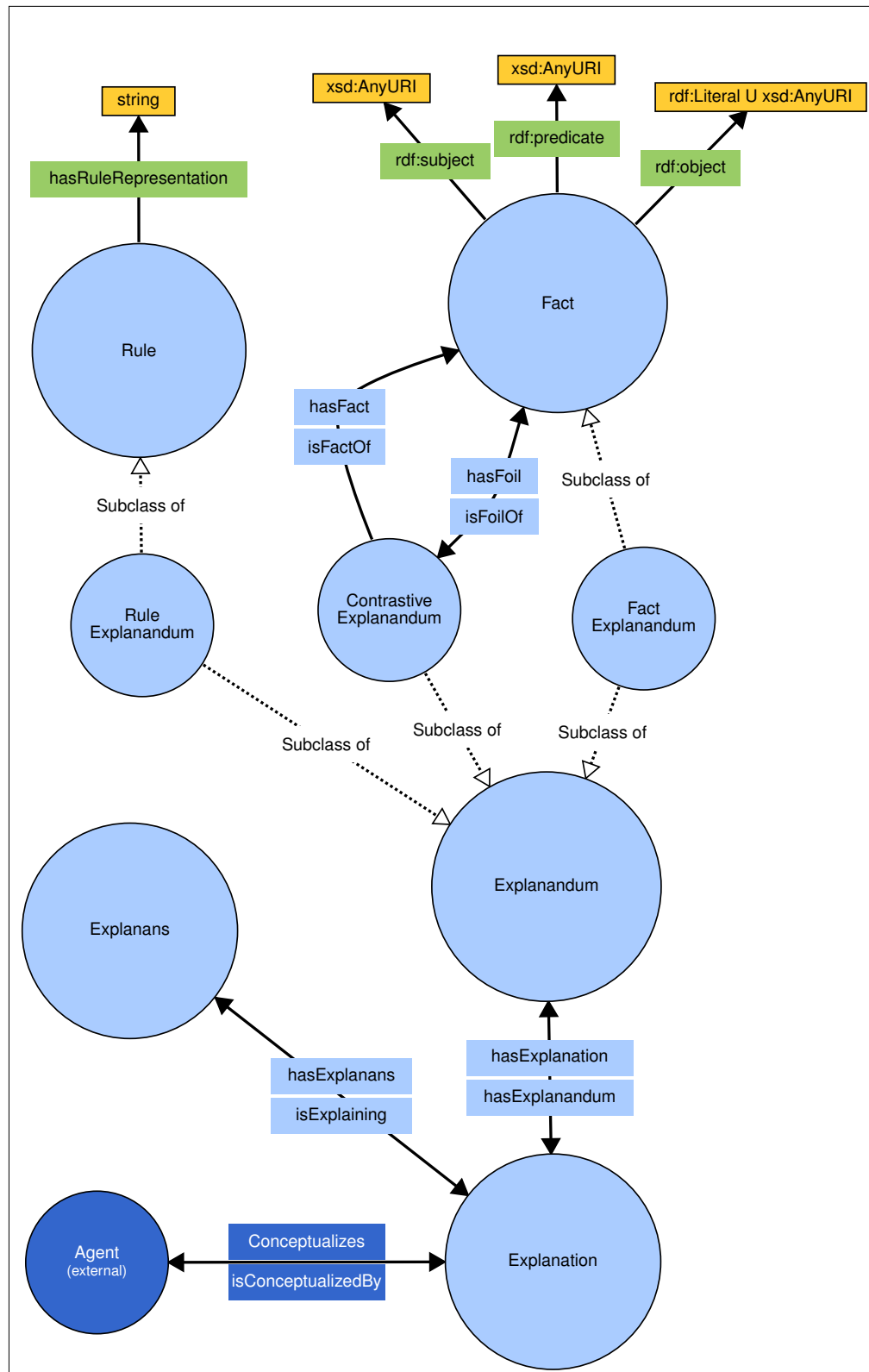


Figure 7.1: Explanandum part of the Explanation ontology module

is internally represented as a Tree instance from the Tree ODP⁵ (Carral et al. 2017). The DecisionRules class is a set of Rule individuals related through the *hasRule* relation. The LinearModel and ModelInspection classes have little internal structure, compared to a tree or a bag of rules. So, the textual or graphical inspection merely has a string datatype containing either the textual inspection itself or a textual representation of the graphical inspection, either as a link or through some encoding.

Because of the size of the OutcomeExplanans we give it its own figure 7.4. Three of its subclasses are bags of items: PrototypeSelection, SaliencyMap, and FeatureImportance. Each bag item is connected via object properties to the relevant bag concepts. The datatype properties *hasSupportingClassification* and *hasStrength* are connected via an intermediate node showing that the union of the three bag items has a role-filler along both datatype properties.

The OutcomeExplanation can also be of a LocalExplanation type. We only modelled a TreePath and a RuleDerivation as local explanation, since these were the main methods we drew from the literature (chapter 2), but this class may be extended by future research. Both the TreePath and RuleDerivation class are instances of a the Tree ODP. A tree path essentially is a unary tree. A rule derivation is a treelike structure showing which combination of facts led the reasoner to derive a particular conclusion.

7.3.2 Mortgage domain ontology

We follow up on the running example of chapter 1 with a dummy data set. Since Plasio internally works on triples, we built a small ontology for the mortgage domain consisting of four main concepts: 1) the mortgage itself, 2) the lender, 3) the borrower, and 4) the property being financed. The ontology is shown graphically in figure 7.5. It consists of the following concepts, datatypes properties, and object properties:

Mortgage: This is the main concept that connects all parts of the mortgage. Each Mortgage instance necessarily has the following properties:

hasApplicationDate: The borrower applied for the mortgage at a particular date and time. The state of the economy corresponding to a mortgage application is checked at this date.

hasConclusionDate: This property contains the date at which a decision is reached on approving the mortgage.

hasInterest: A double value represents the interest rate of the mortgage.

isApproved: A boolean gives the final outcome on approving the mortgage application.

hasBorrower: This object property contains the individual borrowing money from the lender to finance some property.

hasLender: This object property contains the individual that lends money to the borrower.

⁵The OWL file at <http://www.ontologydesignpatterns.org> seems to contain an error. The *TreeNode* class has the expression *not(RootNode)* as an equivalent class. This restriction should be given to the *NonRootNode* instead. Moreover, the OWL file is an implementation of the OWL Full axiomatisation. For our purposes it needs to be brought in OWL DL by applying the changes we described in section 7.2.

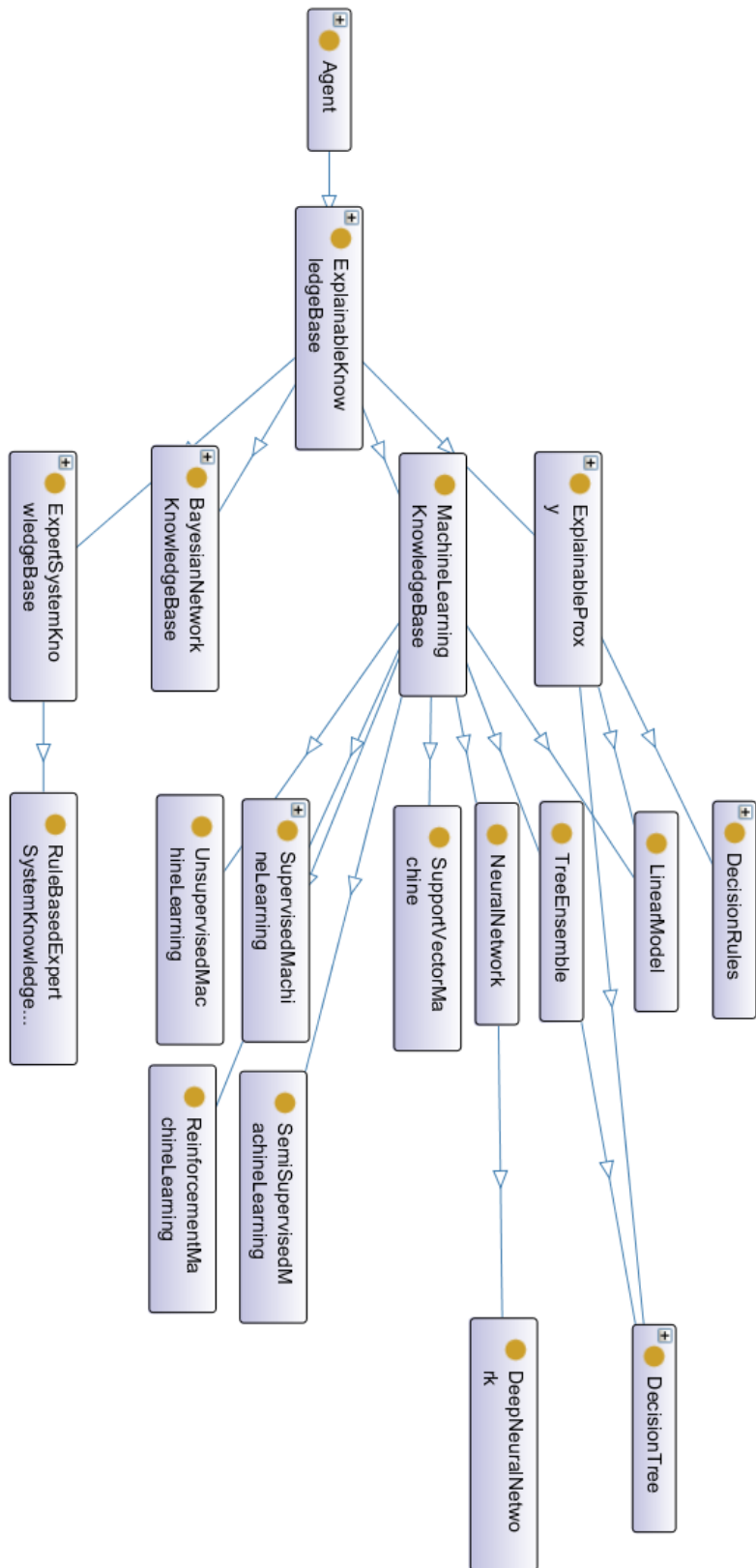


Figure 7.2: Ontology module for knowledge bases visualised using OntoGraf. The arrows represent subclass relations.

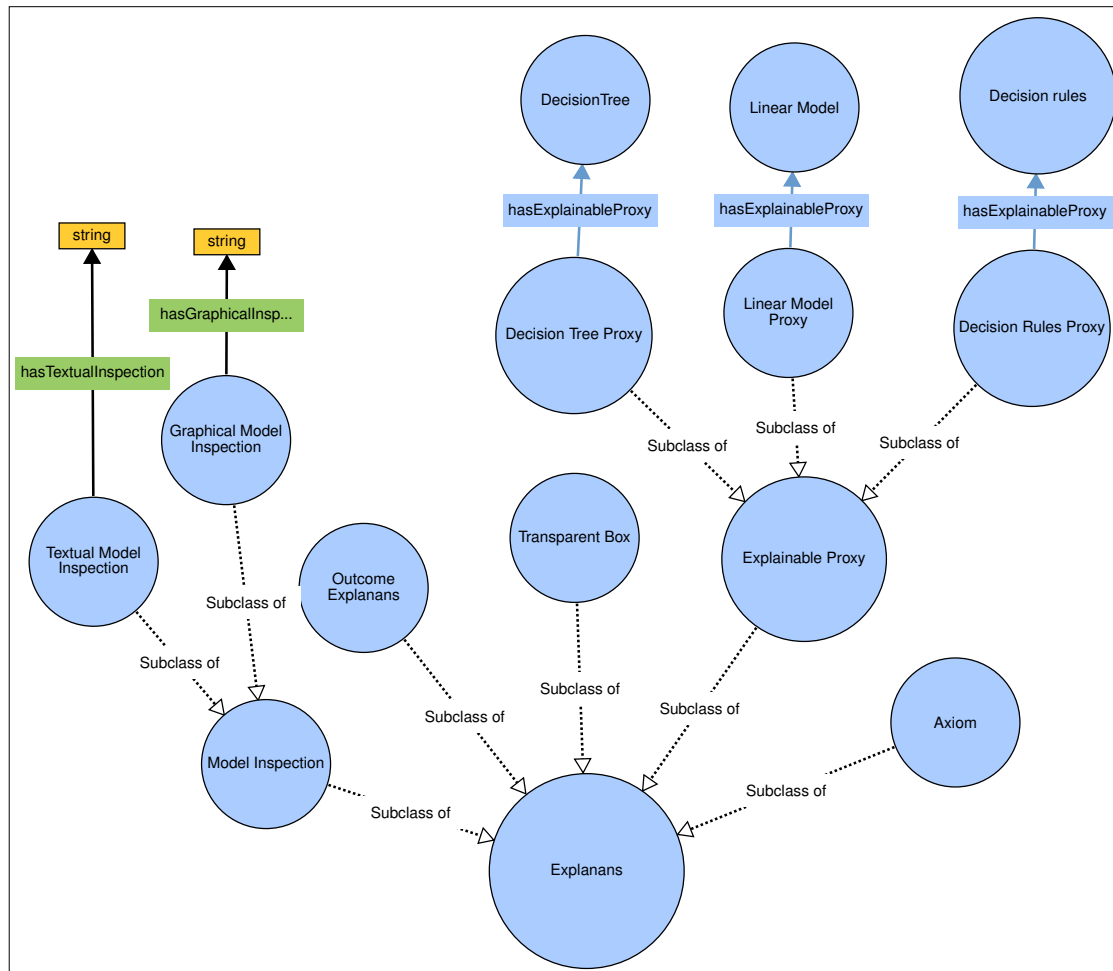


Figure 7.3: Ontology module for the Explanans. The Outcome Explanation concept is further specified in the next figure.

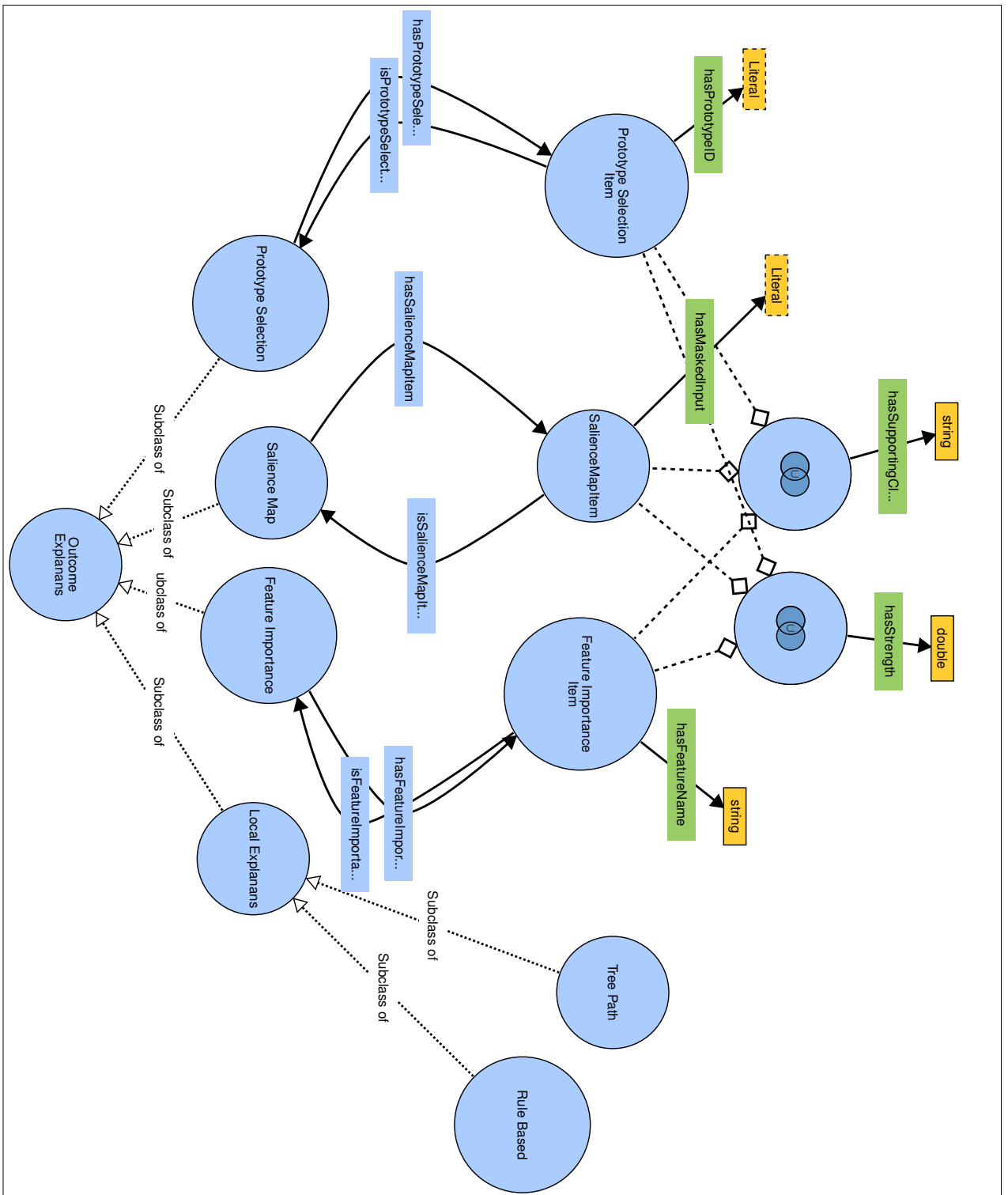


Figure 7.4: Ontology module for an outcome explainers

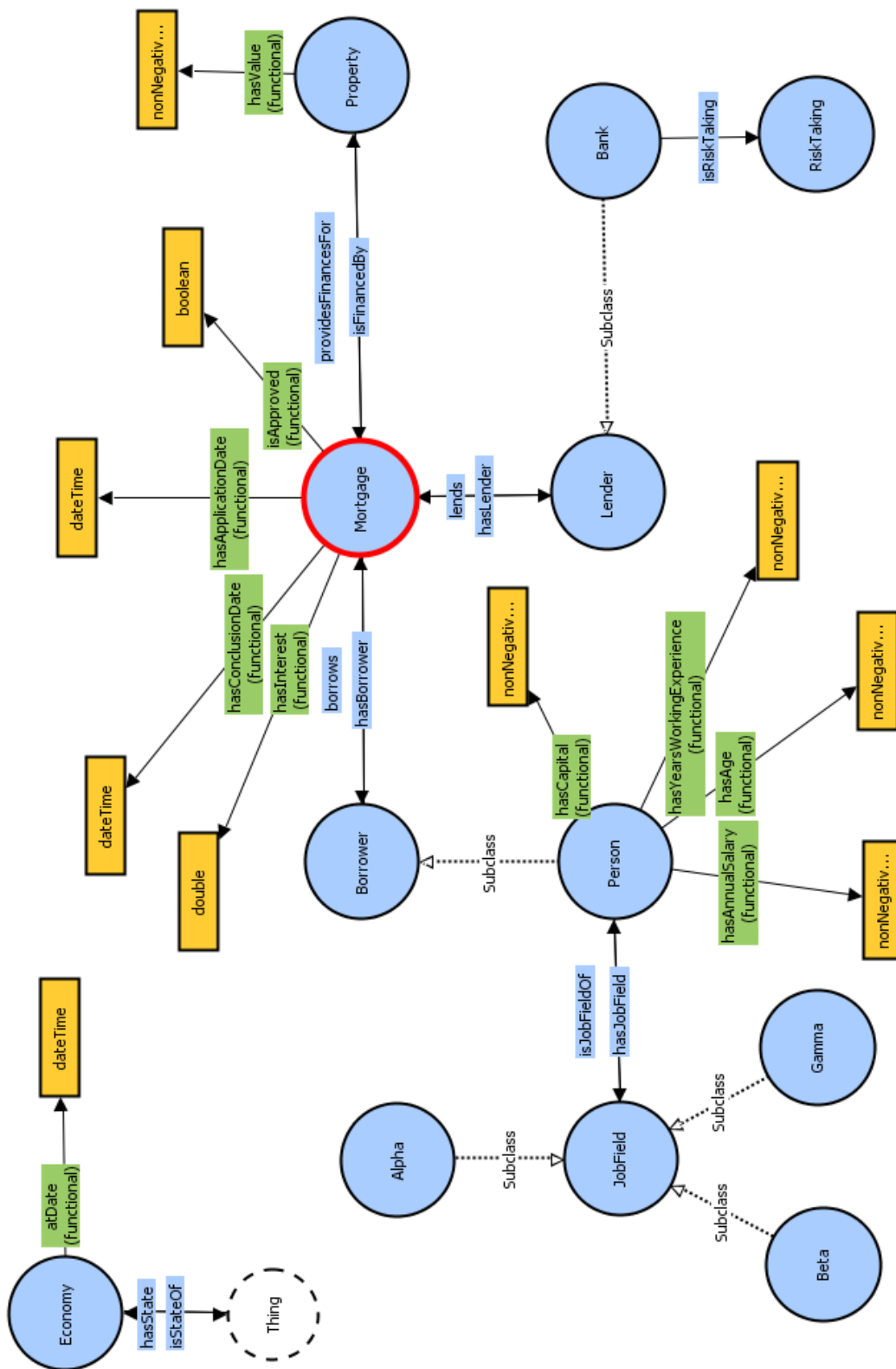


Figure 7.5: Ontology of the mortgage domain

providesFinancesFor: This object property contains the property that is financed by the mortgage.

Property: This concept represents the property that is being financed by the mortgage. It has one data type property:

hasValue: A non-negative integer represents the value of the property the mortgage provides finances for.

Lender: The mortgage has a Lender as the *hasLender*-filler. It represents the entity that provides the loan. In our use case, the Lender concept has one subconcept: Bank. A Bank has a data property RiskTaking related through the *isRiskTaking*-role, which represents an instance of how much risk the bank takes when providing a mortgage for persons who have a higher chance of defaulting.

Borrower: This concept represents the entity that borrows money from the lender to finance some property. In our use case the borrower is a person, who has the following properties:

Age: An integer represents how old the person is.

Annual salary: An integer represents the income a person receives each year.

Capital: An integer represents the value of all property a person owns.

Job field: This object property contains an instance of the job field the person currently works in.

Years working experience: An integer represents the number of years the person has already worked in a particular job field.

Economy: This class contains instances of the current economic state, which can be booming, normal, in a recession, or in a depression. Each instance has an *atDate* relation for each date the economy was in that particular state.

Our data set consists of two Person individuals: 1) Barry and 2) Corrie. The respective properties of both individuals can be found in table 7.2. Please note that the concrete representation of the individuals is in a set of triples, rather than the database format the tables suggest. Both individual have all roles filled, except the *isApproved* role that should contain a boolean value representing the decision. Individual Barry represents a mortgage of a high value, whereas individual Corrie represents a mortgage to finance a property of relatively low value.

7.3.3 ExplanationTree individual

Using both the Explanation ontology and the Mortgage ontology we can construct an ExplanationTree individual. The root Explanation of the individual (figure 7.6) explains the fact that Corrie's mortgage application is rejected. This exact ExplanationTree individual is constructed in the proof of concept by the derivation collection algorithm we will show in chapter 9. So, the individual is an instance of a combined explanation. Each node called "Explanation n ", with n a number, is an instance of the Explanation

	Barry	Corrie
Job field	IT-sector	IT-sector
Annual salary	80.000	35.000
Age	50	25
Years working experience	20	2
Capital	150.000	5.000
Borrows	MortgageHigh	MortgageLow

(a) The two Person individuals

	MortgageHigh	MortgageLow
Interest	5.0	6.0
Conclusion date	2019-03-31	2016-01-01
Application date	2019-01-01	2016-01-01
Provides finances for	House	Apartment
Lender	BankBank	SocialBank
Borrower	Barry	Corrie
Approved	<unknown>	<unknown>

(b) The two Mortgage individuals

	BankBank	SocialBank
Risk taking	Low	High
Lends	MortgageHigh	MortgageLow

(c) The two Bank individuals

	House	Apartment
Value	500.000	170.000
Financed by	MortgageHigh	MortgageLow

(d) The two Property individuals

Table 7.2: The individuals of the dummy data set. The concrete representation of the data is a set of triples: `tnoPocData:Barry tnoPoc:borrows tnoPocData:MortgageHigh`.

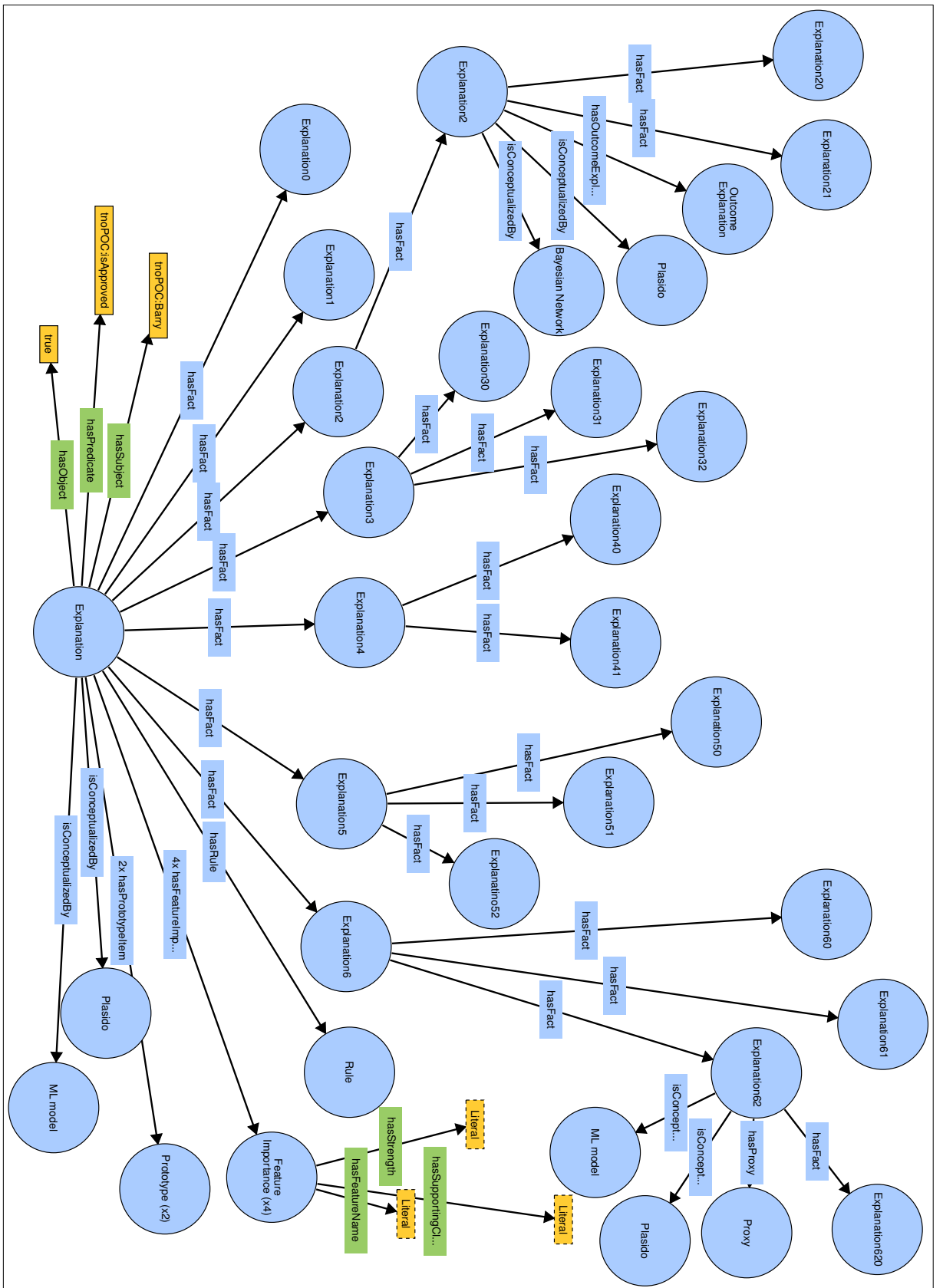


Figure 7.6: Graphical representation of the individual the competency questions are applied on. The numbering of the nodes follows figure 9.1.

ontology module. The complete tree is an instance of the Explanation hierarchy ontology module.

However, the graphical representation is a simplification of the concrete individual. The figure is not large enough to host all classes and relations. Each Explanation node is actually an instance of the Explanandum-Explanation-Explanans combination shown in figure 7.1, where the Explanandum is a Fact individual with its three relations characterising a triple. In the graphical representation only the root node has the necessary relations specifying the fact being explained.

Moreover, each Explanation node has *Plasido* as an *isConceptualisedBy*-filler, which the figure only shows for the root individual, *Explanation2*, and *Explanation620*, which are also conceptualised by external knowledge bases. Moreover, the *hasFact* relation that in the given figure stems from an Explanation, does in fact stem from the Explanans object and has the Explanandum of the Explanation child as object. Without these simplifications it is practically impossible to display all information contained in the combined explanation for a use case of even a moderate size.

The impossibility of visualising an *ExplanationTree* individual that contains all relevant information cannot be held against the ontology. The combined Explanation is not intended to be intelligible through inspection. Already in chapter 4 we introduced competency questions for a dual purpose: to prove that the ontology contains the required information, and to extract the required information from the individual. We defined a set of competency questions to extract the facts from the tree, another to extract the knowledge bases, and a third to cut away parts of the tree that do not contain an external explanation. In the next chapter we will show that the competency questions are sufficient to inspect the *ExplanationTree* individual. This chapter already showed that competency questions are necessary to inspect the tree.

Summary

In this chapter we showed that the Description Logic axiomatisation cannot be directly translated to a corresponding OWL definition. In our case, the axioms defined an ontology that is not decidable and complete. Therefore, we did not implement the formulas that caused the undecidability. We then showed how we did implement the several ontology modules. Our implementation is in the Web Ontology Language (OWL), but we presented it graphically because of the size of the OWL implementation. Our presentation of an *ExplanationTree* individual showed that it contains too much information to be easily visualised. We need competency questions to extract the required information from the *ExplanationTree* individual.

in the correct way, and checks if the ontology correctly implements the requirements that were specified. The competency questions primarily perform the verification part of the evaluation rather than the validation. Another important part of verification is checking whether ontology quality criteria are followed, and if established standards are applied correctly (Falbo 2014, p. 11).

The Ontology Pitfall Scanner² (OOPS!) (Poveda-Villalón et al. 2014) is an example of a method for verification. It performs a set of automated tests, checking the implemented ontology for improper use of relations and annotations, and on failure to follow established conventions. Its results are divided into suggestions and errors, the latter of which are subdivided into critical, important, and minor errors. Our ontology only returns a minor error for some concepts missing a label or comment, where in all these cases the inverse concept does have a label and a comment. It also returns the suggestion of making *hasChild* and *hasParent* symmetric or transitive, but that does not capture the meaning of those relations.

Ontology validation, on the other hand, concerns whether the intended meaning of the elements in the ontology does indeed correspond to the domain that is relevant for the problem. In other words, it checks whether the right ontology was constructed. Some attempts have been made to automate validation of data rich domains (Brewster et al. 2004), but no such data set exists for knowledge base explanation. Therefore, domain knowledge is essential for validation (Falbo 2014). An instantiation or graphical representation of the ontology becomes an important aid, since domain experts usually are not ontology experts. However, in our study we perform the role of both domain expert and ontology expert. Nevertheless, we present an instantiated Explanation individual and a proof of concept (chapter 9) such that the reader who we do not expect to be an ontology expert, can to some extent validate the ontology.

8.2 SPARQL

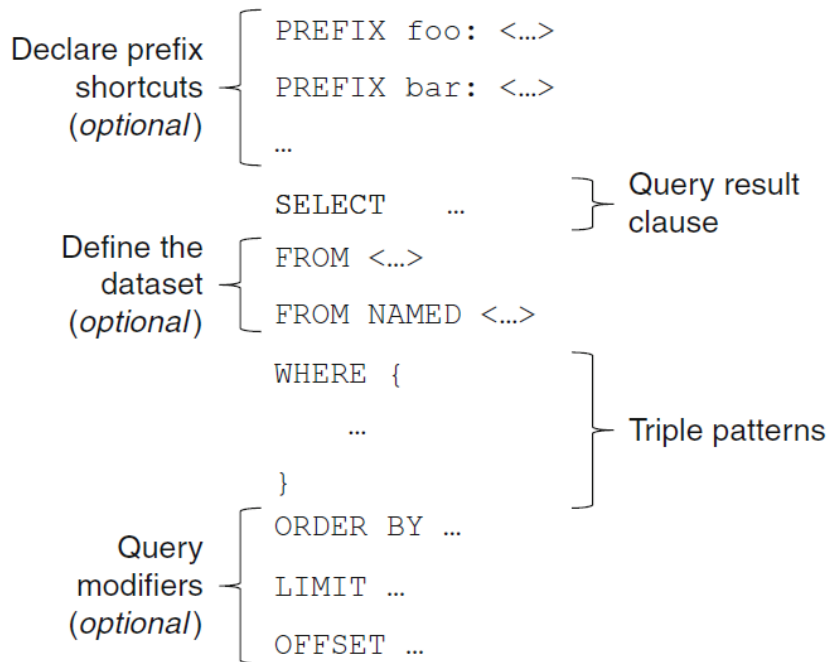
We ask questions to the OWL knowledge base through the SPARQL Protocol And RDF Query Language (SPARQL). We show the syntax in figure 8.1. SPARQL explicitly does not function as a reasoner. It merely looks for the existence of a triple pattern in the graph representation. Horridge and Musen (2016) developed Snap SPARQL as a query language with reasoning capabilities, but its syntax is not expressive enough for our purposes.

The **SELECT**-clause is the heart of the query. It specifies the triple pattern a graph node should contain to be returned. The **SELECT** query returns its answer as a set of variable bindings. We also use a **CONSTRUCT** clause when we want to structure the data.

SELECT returns a table with for each possible answer a row entry containing the variable bindings. Query A.1 (in the appendix) gives a simple example.

CONSTRUCT returns its answer as a set of triples forming a graph instance. The first step tries to find a set of variable bindings following the **WHERE** pattern, similar to a **SELECT** query. However, when a set of valid variable bindings is found, the **CONSTRUCT** pattern is instantiated with the variable bindings. So, the result of

²<http://oops.linkeddata.es/>



Listing 8.1: Syntax of a SPARQL query (Della Valle and Ceri 2011, fig. 8.2)

the query is the combined set of triples for each valid set of variable bindings. Query A.7 is an example that uses the `CONSTRUCT` operation.

8.3 Competency questions

In this section we will for each ontology module give the result of the SPARQL implementation for each competency question. We will first test the competency questions of the Explanation hierarchy module. We will then focus on the Explanation module and the Knowledge Base module. This way we start with a broad view which becomes more detailed with each applied competency question. The questions are applied on the `ExplanationTree` individual given in figure 7.6. This `ExplanationTree` individual contains too much information for its correctness to be quickly observed or for its contents to be extracted by visual inspection. So, the competency question play a double role of showing the correctness and extracting information.

8.3.1 Explanation hierarchy

We constructed three competency questions for the explanation hierarchy module. The first query extracts the facts from the `ExplanationTree`, while discarding information about the conceptualising knowledge bases and the explanantia. The second query isolates the conceptualising KBs. The third query puts its focus on the Explanans objects containing an explanation extracted from a machine learning KB or Bayesian network. The three competency questions together show the `Explanation Tree` to contain complete information about the facts, the knowledge bases, and the explanantia.

A visual representation of the output of the competency question extracting the fact tree is shown in figure 8.2. Each node is labelled with a description of the fact it contains. The formal output contains for each node the URI uniquely identifying the Explanandum individual containing the triple fact, allowing the competency question extracting a fact from a single explanation to be consecutively applied. This competency question will be covered in section 8.3.2.

The question extracting the knowledge bases produces a tree with a similar structure (figure 8.3). However, this instance does not contain the facts, but contains the conceptualising knowledge bases, such that it becomes clear which part of the explanation consulted an external source. In the example output we coloured the Explanations that extract an explanation from an external source. The concrete implementation gives each Explanation node a label for the knowledge bases that produced the explanation. We cannot use the exact structure as in the previous competency questions, because a single KB may conceptualise more than one Explanation.

The last ExplanationTree competency question produces a tree containing only the nodes that drew an explanation from an external knowledge base. Its output is graphically shown in figure 8.4. Several Explanation nodes in the example individual (figure 7.6) are constructed by the ontology reasoner, because the reasoner has to infer, for example, the value of the property a person wants financed by the mortgage. That triple does not exist in the knowledge base, as the lack of a direct relation in the graphical representation of the mortgage ontology (figure 7.5) shows. The derivation produced by the inference is included in the Explanation individual for completeness sake, but in most use cases the explanation of the external source is more important.

The competency question therefore selects the Explanations conceptualised by an external knowledge base. It adds parent-child relations to the appropriate nodes, which may not have had child-parent relations in the original individual. A parent-child relation may be added to nodes that were originally connected through an intermediate node which was cut because it did not have an Explanation drawn from an external knowledge base. In the example individual (figure 7.6) node *Explanation6* connects the root node with node *Explanation62*. The output of the competency question (figure 8.4) shows that the intermediate node is replaced by a *hasChild* relation. The output also closely resembles the graphical representation of the use case (figure 1.1), showing that the intuitive configuration of the external knowledge bases can be extracted from the explanation.

8.3.2 Fact explanation

The Explanation module competency questions allow us to zoom in on a single Explanation individual instead of an ExplanationTree individual. These competency questions extract the fact being explained, the type of explanation, the explainable parts, and the conceptualising knowledge from the Explanation individual. The results do not exhibit a treelike structure, so we use SELECT queries instead of CONSTRUCT queries.

We present the output for the root Explanation individual as an example, but the competency question can similarly be applied to the other nodes of the tree. Table 8.1 shows the triple being explained in the root node. The contents of the Explanans are divided among two competency questions. Table 8.2 gives the type of Explanans fol-

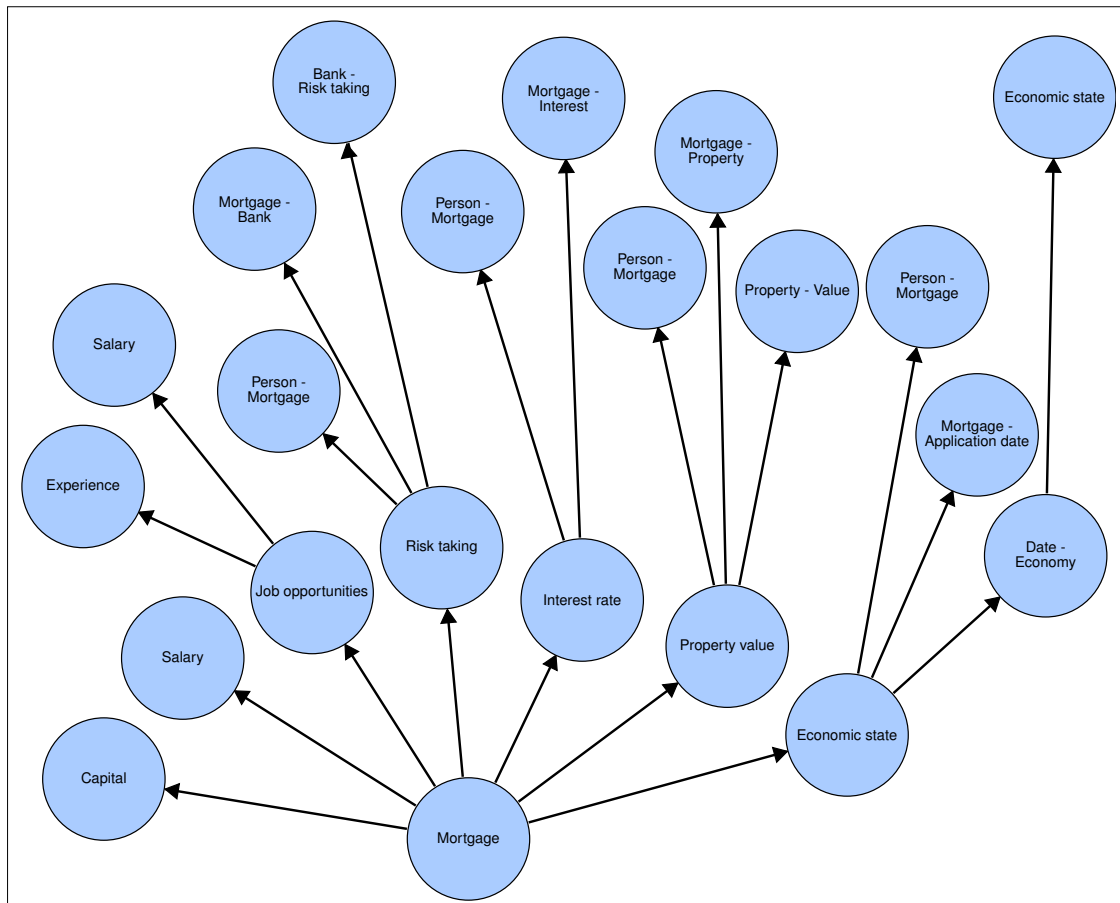


Figure 8.2: Visual representation a simplified fact tree corresponding to SPARQL query A.7. Each arrow represents a *hasChild* relation.

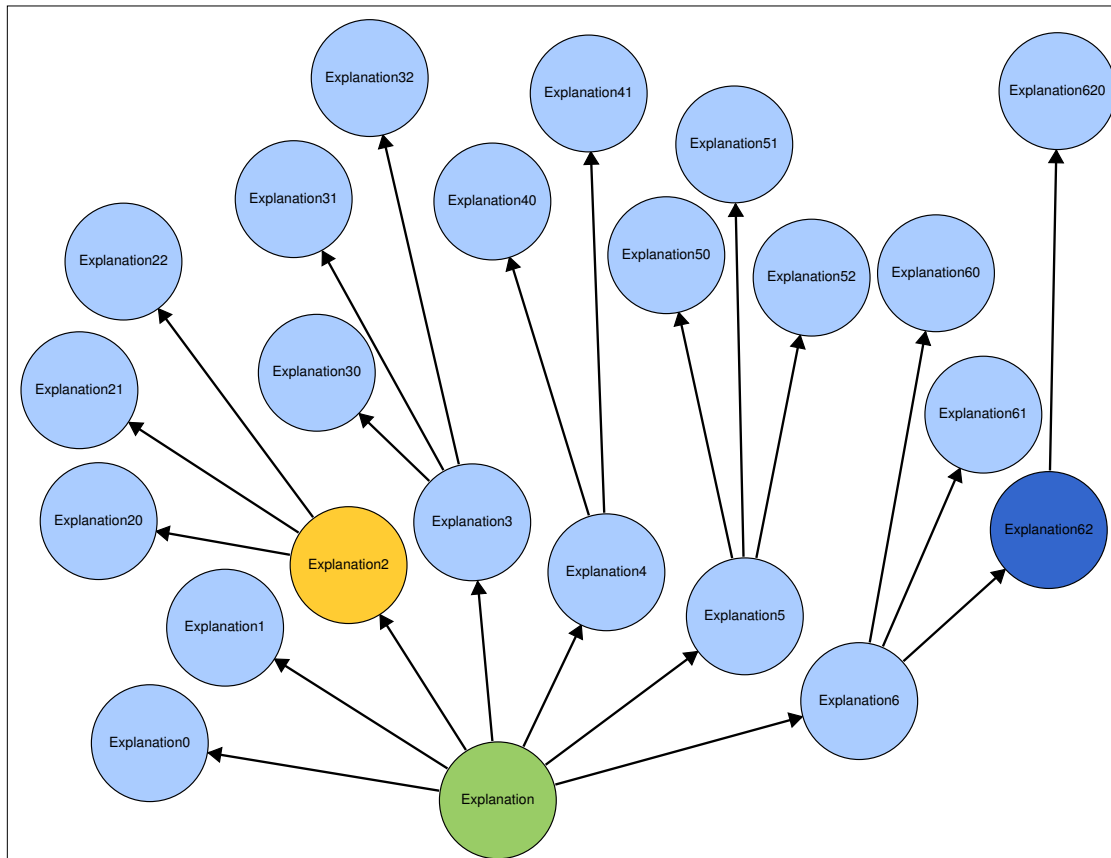


Figure 8.3: Visual representation of the simplified knowledge base tree corresponding to SPARQL query A.9. The mortgage explanation, job opportunities explanation, and economy explanation are colored green, yellow, and dark blue, respectively, because these explanations are conceptualised by external KBs. Each arrow represents a *hasChild* relation.

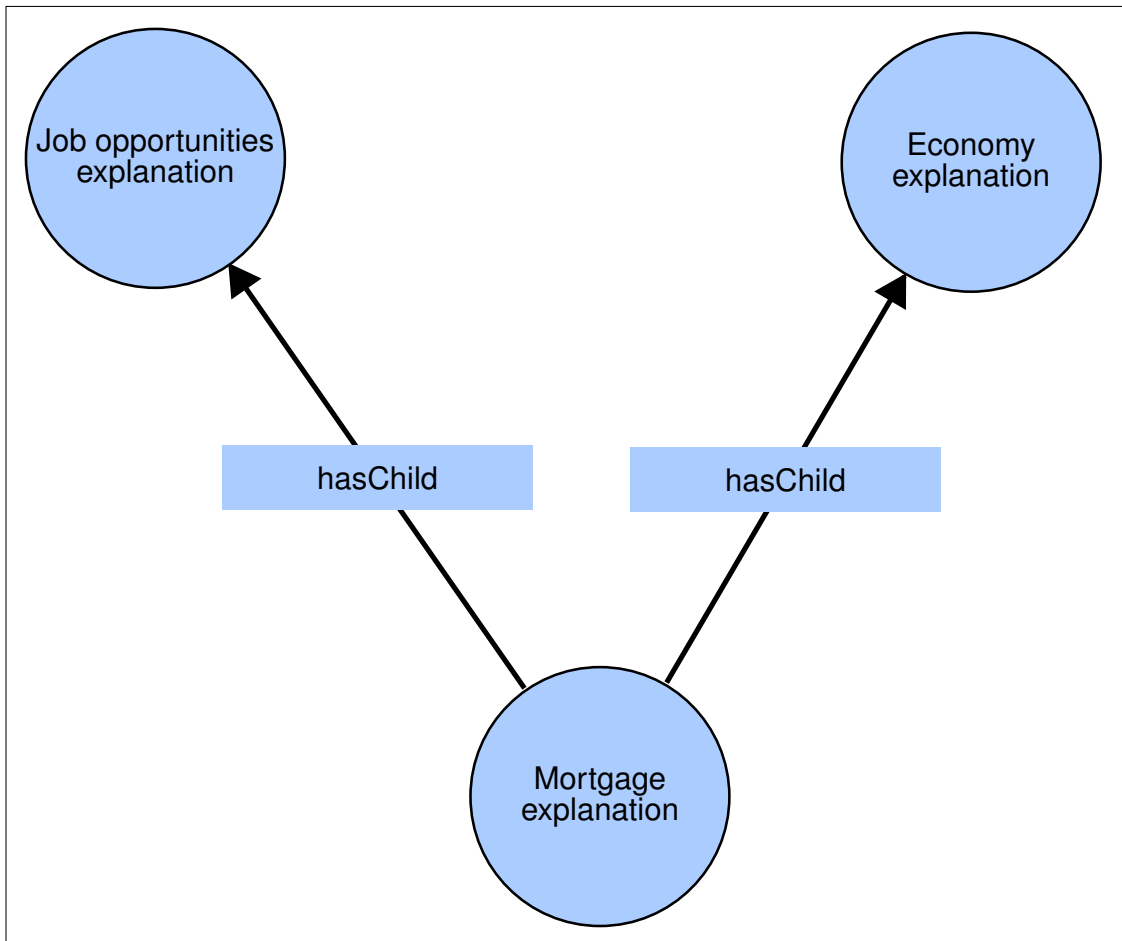


Figure 8.4: A visual representation of the simplified explanation tree corresponding to the output of SPARQL query A.8. Each explanation node also has access to its related Explanans, Explanandum, and KnowledgeBase.

?explanation	?subject	?predicate	?object
tnoData:ExplanationIndividual	tnoPocData:Corrie	tnoPoc:isApproved	false

Table 8.1: Which fact is being explained? Sample output of SPARQL query [A.1](#) applied on the root node of the ExplanationTree individual.

?explanation	?type
tnoData:ExplanationIndividual	tno:OutcomeExplanationExplanans

Table 8.2: What is the role of the explanation? Sample output of SPARQL query [A.2](#) applied on the root node of the ExplanationTree individual.

lowing the categorisation we constructed in chapter [2](#). The contents of the Explanans are extracted as shown in table [8.3](#). In chapter [4](#) we stated the competency questions per type of Explanans, but the definition of *hasExplainablePart* in chapter [6](#) allows us to generalise the competency questions to a single SPARQL query.

The results of the fourth competency questions are shown in table [8.4](#). It returns only the unique URI of the knowledge base. A specification of the knowledge base can be obtained through the competency questions for the knowledge base module.

8.3.3 Knowledge bases

The SPARQL implementations of the competency questions for the knowledge base module are simpler than the other queries, because the knowledge base module uses very little reasoning. It is more of a taxonomy containing only subclass relations. So, both SPARQL queries merely ask which concepts the KnowledgeBase individual belongs to.

The first competency question returns which general subtypes of knowledge bases the individual belongs to. It only returns whether the KnowledgeBase is a type of machine learning model, Bayesian network, or expert system. The second competency question gives a more detailed look by returning all concepts the individual belongs to. It only excludes trivial results such as *owl:Thing* and *dul:Agent*. The former is trivial, because it is the root concept that any custom concept necessarily belongs to. The latter is also trivial, because it is the range of the *isConceptualizedBy* relation. The output is given in table [8.5](#) and table [8.6](#) respectively. Both tables show the results for all KnowledgeBase individuals in our use case.

Summary

In this chapter we presented the competency questions formalised in SPARQL, thereby showing that both detailed facts as well as tree summaries can be extracted from an ExplanationTree individual. In the next section we will give our proof of concept, describing how the Plasido framework constructs an ExplanationTree individual. The SPARQL queries described in this chapter are applicable on any ExplanationTree individual constructed by Plasido.

?explanation	?part
tnoData:ExplanationIndividual	tnoData:MortgageClassificationPrototypeSelectedItem1-
tnoData:ExplanationIndividual	tnoData:MortgageClassificationPrototypeSelectedItem2-
tnoData:ExplanationIndividual	tnoData:MortgageClassificationFeatureImportanceItem1-
tnoData:ExplanationIndividual	tnoData:MortgageClassificationFeatureImportanceItem2-
tnoData:ExplanationIndividual	tnoData:MortgageClassificationFeatureImportanceItem3-
tnoData:ExplanationIndividual	tnoData:MortgageClassificationFeatureImportanceItem4-
tnoData:ExplanationIndividual	tnoData:RuleIndividual
tnoData:ExplanationIndividual	tnoData:ExplanandumIndividualLeaf0
tnoData:ExplanationIndividual	tnoData:ExplanandumIndividual1
tnoData:ExplanationIndividual	tnoData:ExplanandumIndividual2
tnoData:ExplanationIndividual	tnoData:ExplanandumIndividual3
tnoData:ExplanationIndividual	tnoData:ExplanandumIndividual4
tnoData:ExplanationIndividual	tnoData:ExplanandumIndividual5
tnoData:ExplanationIndividual	tnoData:ExplanandumIndividualLeaf6

Table 8.3: Which items does the the explanation consist of? Sample output of SPARQL query A.3 applied on the root node of the ExplanationTree individual.

?explanation	?knowledgeBase
tnoData:ExplanationIndividual	tnoData:PlasidoKnowledgeEngine
tnoData:ExplanationIndividual	tnoData:ExplainableMortgageMachineLearningKB

Table 8.4: Which KB is the explanation derived from? Sample output of SPARQL query A.4 applied on the root node of the ExplanationTree individual.

?knowledgeBase	?type
tnoData:ExplainableJobOpportunitiesBayesianNetworkKB	tno:BayesianNetworkKnowledgeBase
tnoData:PlasidoKnowledgeEngine	tno:ExpertSystemKnowledgeBase
tnoData:ExplainableEconomyMachineLearningKB	tno:MachineLearningKnowledgeBase
tnoData:ExplainableMortgageMachineLearningKB	tno:MachineLearningKnowledgeBase

Table 8.5: Which type among expert system, ML model, and Bayesian network does the KB belong to? Output of SPARQL query A.5 applied on the complete ExplanationTree individual.

?knowledgeBase	?type
tnoData:PlasidoKnowledgeEngine	tno:ExpertSystemKnowledgeBase
tnoData:PlasidoKnowledgeEngine	tno:RuleBasedExpertSystemKnowledgeBase
tnoData:PlasidoKnowledgeEngine	tno:ExplainableKnowledgeBase
tnoData:ExplainableJobOpportunitiesBayesianNetworkKB	tno:BayesianNetworkKnowledgeBase
tnoData:ExplainableJobOpportunitiesBayesianNetworkKB	tno:ExplainableKnowledgeBase
tnoData:ExplainableEconomyMachineLearningKB	tno:MachineLearningKnowledgeBase
tnoData:ExplainableEconomyMachineLearningKB	tno:ExplainableKnowledgeBase
tnoData:ExplainableMortgageMachineLearningKB	tno:SupervisedMachineLearning
tnoData:ExplainableMortgageMachineLearningKB	tno:MachineLearningKnowledgeBase
tnoData:ExplainableMortgageMachineLearningKB	tno:ExplainableKnowledgeBase

Table 8.6: Which combination of subtypes identifies the KB? Output of SPARQL query A.6 applied on the complete ExplanationTree individual.

Chapter 9

Proof of concept

TNO is a research institution that operates at the intersection of scientific research and business products. Especially the Connected Business group, where the internship took place, provides, as the name suggests, corporate interoperability solutions. Therefore, a proof of concept (POC) was expected as part of the deliverables. In this section we will present the implementation of the POC to prove that our ontology can function as intended. Since our POC adds explanation functionality to Plasido we will call it Explainable Plasido.

First, we make the reader familiar with the architecture of Plasido (Nouwt 2016), the framework developed at TNO to link various data sources. Plasido already makes extensive use of functionality provided by Apache Jena¹, which is a large open source framework for semantic web and linked data applications. We want Explainable Plasido to function as a plugin to Apache Jena instead of a Jena branch. So, we were forced to work with the methods and classes already provided. This did not influence the behaviour of the POC, but did make some functions to be less elegantly implemented than if we would have changed the Apache Jena source.

This chapter describes our additions to Plasido, which can be divided into two categories: 1) generic and 2) domain specific. The generic parts ought to be reused in a future use case. The domain specific parts are intended as a model to be followed in a future use case. In the last section we describe the flow of control when querying Explainable Plasido on a sample input.

9.1 Explainable Plasido

The current version of Plasido can only combine data from various outside sources, but has no functionality to either draw or combine explanations from the external sources. The communication of Plasido is performed by a reasoner operating on a knowledge base consisting of a set of facts and a rule base. In our proof of concept the collection of facts is a triple representation of table 7.2. The set of symbolic rules $(p_0, \dots, p \rightarrow q)$ defines how to derive a previously unseen fact, possibly involving calling an external knowledge base. The communication to an external source is performed by a so-called Builtin, which is as an element of the body of a rule. Nouwt and Verhoosel (2018) showed how

¹<http://jena.apache.org>

Apache Jena Builtins can be leveraged to draw data from an external KB. We extended the Builtin such that we can also query the explanation of the returned value. Each explanation collected this way is a partial explanation. These are then combined by a derivation collector and parsed into an individual of the ExplanationTree ontology.

9.1.1 Generic

Our additions contain three general classes that should be reused in a future use case: 1) The DerivationCollector takes Apache Jena's trace of which symbolic rules were followed to derive a particular triple, and parses it into an instance of the ontology. During the construction of the ontology instance, the Builtins are called for an explanation. 2) A general abstract class DerivationBaseBuiltin defines the form of an explainable external source. The construction of an abstract class allows the instantiations to inherit common methods. It also allows the references in the DerivationCollector to be handled at runtime by Java *polymorphism*. 3) A DerivationService class manages the flow of control. We explain these classes in more detail.

DerivationCollector: This function takes as its input a trace of the rules it followed to derive a certain fact. Each rule is turned into an Explanation instance. During this process each external KB is called to provide an explanation for its output.

We assume that the explanation provided by the external KB already is in the format defined by our ontology. After all, the goal of our thesis is not to define an automatic transformation into the language of the ontology. Writing the explanation of the external KB remains a task to be done manually, although an automation procedure can be constructed such that construction of the explanation instance can be done semi-automatically. To be more precise, the DerivationCollector should receive from the external KB an Explanans instance rather than a complete Explanation instance.

During construction we give each node a uniquely identifying URI. If two individuals have the same URI, an ontology reasoner regards both as identical. So, we have to ensure that two nodes have a different URI unless we are certain both are identical. To uniquely identify a tree node we use a suffix of n digits that is unique for each position in the tree. The length of the suffix gives the depth. Digit c at index i of the suffix means that we take the c 'th child at depth i . The root node has the empty suffix. Figure 9.1 gives an example.

DerivationBaseBuiltin: Plasido uses the Builtins provided by Apache Jena to connect with an external KB. The Jena framework provides an abstract BaseBuiltin which can call an outside KB with a set of variables, retrieve an output, and bind it to an uninstantiated variable (Nouwts and Verhoosel 2018). The Builtin we use to connect to an external explainable KB must provide additional functionality. First of all, it should be able to return an explanation for the given input. Secondly, it should return the KB the explanation is derived from.

To perform the additional functionality we programmed a DerivationBaseBuiltin (listing 9.3). This abstract class provides methods to receive an instance of the KB in the format of the ontology and an instance of the explanation in the format of

```

1 # Rules using Builtins to connect to an external KB
2 [economyRule:
3     (?mortgage tnoPOC:hasApplicationDate ?dateTime),
4     ExplainableEconomyPredictor(?dateTime, ?economicState) ->
5     (?economicState tnoPOC:atDate ?dateTime) ]
6 [opportunitiesRule:
7     (?person tnoPOC:hasYearsWorkingExperience ?yearsWE),
8     (?person tnoPOC:hasAnnualSalary ?annualS),
9     (?person tnoPOC:hasJobField ?jobF),
10    ExplainableJobOpportunitiesPredictor(?yearsWE,
11        ?annualS, ?jobF, ?output) ->
12    (?person tnoPOC:hasJobOpportunities ?output) ]
13 [mortgageRule:
14    (?person tnoPOC:hasCapital ?capital),
15    (?person tnoPOC:hasJobOpportunities ?promotions),
16    (?person tnoPOC:getPropertyValue ?propertyValue),
17    (?person tnoPOC:getInterestRate ?interestRate),
18    (?person tnoPOC:getEconomicState ?economicState),
19    (?person tnoPOC:getBankRiskTaking ?riskTaking),
20    (?person tnoPOC:hasAnnualSalary ?annualSalary),
21    ExplainableMortgagePredictor(?capital,?promotions,
22        ?propertyValue,?interestRate,?economicState,
23        ?riskTaking,?annualSalary,?result) ->
24    (?person tnoPOC:isApproved ?result) ]

```

Listing 9.1: These rules direct the flow of knowledge through the network. Each rule contains a Builtin as the last item of the body. Internally, the arguments of the Builtin are passed on to request a value from an external KB. The return value is bound to the last parameter of the Builtin, which is then unified with the last parameter of the head. This way we obtain new knowledge from the Builtin and add it to the knowledge base. The other triples in the body of the rules are used to draw existing data from the knowledge base. For example, the mortgage classification ML model takes the seven values indicated by the Builtin parameters. The data is drawn directly from the knowledge base or an intermediate rule from listing 9.2.

```

1 # Rules extracting data from the Mortgage domain ontology.
2 [getPropertyValue:
3     (?mortgage tnoPOC:hasBorrower ?person),
4     (?mortgage tnoPOC:providesFinancesFor ?property),
5     (?property tnoPOC:hasValue ?propertyValue) ->
6     (?person tnoPOC:getPropertyValue ?propertyValue) ]
7 [getEconomicState:
8     (?mortgage tnoPOC:hasBorrower ?person),
9     (?mortgage tnoPOC:hasApplicationDate ?dateTime),
10    (?economicState tnoPOC:atDate ?dateTime) ->
11    (?person tnoPOC:getEconomicState ?economicState) ]
12 [getInterestRate:
13    (?mortgage tnoPOC:hasBorrower ?person),
14    (?mortgage tnoPOC:hasInterest ?interestRate) ->
15    (?person tnoPOC:getInterestRate ?interestRate) ]
16 [getBankRiskTaking:
17    (?mortgage tnoPOC:hasBorrower ?person),
18    (?mortgage tnoPOC:hasLender ?bank),
19    (?bank tnoPOC:isRiskTaking ?riskTaking) ->
20    (?person tnoPOC:getBankRiskTaking ?riskTaking) ]

```

Listing 9.2: These rules extract data from the Mortgage domain ontology. These are necessary because the triple representation of the mortgage ontology (figure 7.5) only contains a concrete triple for direct relations. For example, figure 7.5 shows that a Person does not have a role filled by a Property. Rather, the Property corresponding to a person's mortgage application can only be reached via the Mortgage. The steps from Person to Mortgage, Mortgage to Property, and Property to a literal representing the property value are exactly the three triples in the body of the `getPropertyValue`-rule.

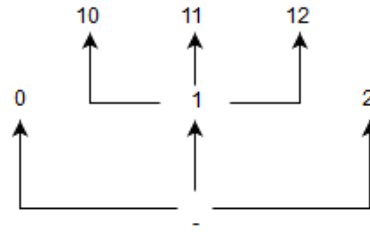


Figure 9.1: Use of suffixes to uniquely define the position in a tree

the ontology. An instance of the abstract class has to be constructed for each Builtin, which we will show in the next section on domain specific parts of the proof of concept.

DerivationService: This class reads the triple queried by the user and searches the knowledge base for an output value. It also gives the derivation tree to the DerivationCollector class to parse the derivation into an instance of the ontology.

```

1  abstract class DerivationBaseBuiltin extends BaseBuiltin {
2      abstract Model getExplanation(
3          Node[] builtinVariables ,
4          int argLength ,
5          Node[] builtinInstantiatedVariables ,
6          String explanansUri ,
7          String suffix );
8
9      abstract Model getKnowledgeBaseModel ();
10
11     abstract Resource getKnowledgeBaseModel ();
12 }
    
```

Listing 9.3: Abstract DerivationBaseBuiltin class

9.1.2 Domain specific

Three parts are domain-specific. We implemented these for our proof of concept to work properly, and as an example for future use cases.

Reasoner rules: The rules in Plasido guide the output of a KB to the input of another KB. The hierarchy of the KBs is configured in the rule base. The Plasido syntax is quite broad with support for both backward reasoning and forward reasoning to increase efficiency. We are currently not optimizing for efficiency, so we restrict the syntax as shown in listing 9.4. In particular, we eliminate backward reasoning and only allow rules with one head. These restrictions do not decrease expressivity.

The rules we define for the use case can be divided into 1) connecting to an outside source and 2) gathering data from the triple store. Listing 9.1 gives the rules where

```

1 Rule      := [ ruleName : bare-rule ]
2
3 // forward rule
4 bare-rule := term, ..., term -> term
5
6 // triple pattern or invoke a builtin
7 term      := (node, node, node)
8            or builtin(node, ..., node)
9
10 node     := uri-ref           // e.g. http://foo.com/eg
11            or prefix:localname // e.g. rdf:type
12            or <uri-ref>       // e.g. <myscheme:myuri>
13            or ?varname        // variable
14            or 'a literal'     // a plain string literal
15            or 'lex'^^typeURI  // a typed literal
16            or number          // e.g. 42 or 25.5

```

Listing 9.4: The restricted rules syntax for Explainable Plasio

an external KB is consulted. Typically, we define one rule per external KB, where the other clauses of the rule extract data from the triple store. In listing 9.2 the rules are given which do not connect to an outside source, but collect triples that are not explicitly stored in the knowledge base. For example, we first need to get the associated mortgage to obtain the value of the property of a particular borrower. In the concrete graph representation of the knowledge, we need to follow the edges connecting the borrower to the mortgage, the mortgage to the property, and the property to the integer connected via the *hasValue* role.

Knowledge bases connected through a proxy server: In a proper use case all data sources are connected through endpoints on the web. We simulate this situation by setting up a local server, which provides for all `DerivationBaseBuiltin` instances an endpoint where both a value and an explanation can be requested.

Implementation of the abstract `DerivationBaseBuiltin`: The implementation of the `DerivationBaseBuiltin` queries the internet location for an outcome value and a corresponding explanation. This location is explicitly allowed to be an internet location for interoperability purposes. That way, the explanation service can link to data sources at other organisations.

In our proof of concept, we construct three instantiation of the abstract `DerivationBaseBuiltin` class:

ExplainableEconomyBuiltin: This Builtin returns as explanation an instance of a rule-based proxy approximating the reasoning of the ML model underlying the classification. Its rules are shown in listing 9.5.

ExplainableJobOpportunitiesBuiltin: As the measure of job opportunities is computed by a BN, this Builtin returns an instance of Timmer et al. (2017)'s support graph, shown in figure 9.2.

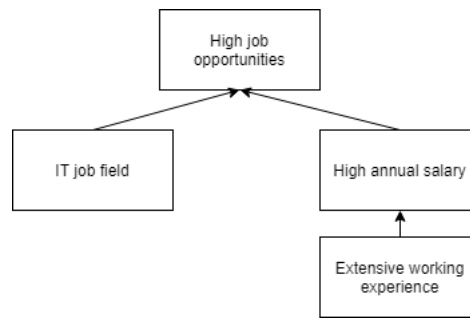


Figure 9.2: Instance of the support graph for job opportunities explanation

ExplainableMortgageBuiltin: The explanation of the mortgage classification consists of both feature importance items and prototypes selected from the training set. It consists of four feature importance items and two prototypes items. This explanation is an example of what would be outputted by Adhikari (2018)'s LEAFAGE method.

```

USA_Recession <- Dow_Jones < 15.000
USA_Booming <- Dow_Jones > 25.000
NL_Recession <- AEX < 450
NL_Blooming <- AEX > 550

Depression <- USA_Recession ^ NL_Recession
Recession <- USA_Recession
Booming <- USA_Booming ^ NL_Booming
Normal <- ~Depression ^ ~Recession ^ ~Booming
    
```

Listing 9.5: Rule-based proxy of the economic state classifier

9.2 Flow of control

Let us ask Explainable Plasido whether individual Barry's mortgage application is approved. Formally, we query whether the following triple is contained in the knowledge base:

```
tnoPoc:Barry tnoPoc:isApproved "false"^^xsd:boolean
```

The first step is the initialisation procedure. Explainable Plasido combines its facts (i.e. the data about individuals Barry and Corrie) with the rules shown in listings 9.1 and 9.2. All derivations are computed at the initialisation step by using *forward reasoning*, instead of using *backward reasoning* which would derive only facts relevant to the query. Forward reasoning decreases reasoning times for further queries. All Derivation-Builtins contained in the rules are queried for an output value, although not yet for an explanation. For each new fact its derivation sequence is kept track of.

After deriving all facts from the knowledge base, it is checked whether the queried fact is among those. If true, its derivation sequence is given to the DerivationCollector, which parses the sequence into an instance of the Explanation ontology. The Derivation object produced by Apache Jena contains which rules were followed to arrive at a particular conclusion, so it only contains a rule-based explanation. The explanations from the explainable Builtins are requested when a rule is found with a Builtin in its body. The Builtins connect to a local server that provides the explanation as Explanans instances. The DerivationCollector unifies the Explanans it receives from the Builtin with the Explanans in the ExplanationTree by giving both the same unique suffix identifying their position in the tree.

Finally, we need to apply a reasoner on the ExplanationTree individual. The DerivationCollector constructs the tree with just the necessary roles. An ontology reasoner adds inferred relations, such as additional ancestor or descendant relations, and inverse relations. For this goal we employ the HermiT reasoner (Glimm et al. 2014; Shearer et al. 2008). The competency questions can be applied to the inferred tree.

Summary

In this chapter we outlined the explanation functionality that makes up Explainable Plasido. Our main addition is an explanation combination algorithm that takes as its input a derivation trace provided by Apache Jena. It turns the trace into an ExplanationTree structure, while simultaneously collecting explanations from the external knowledge bases indicated by the trace. We also provided an implementation of the domain specific parts of Explainable Plasido.

Chapter 10

Conclusion

In this thesis we proposed an ontology for the combination and communication of explanations in a federated knowledge environment. In this chapter we will formulate an answer to the research questions, ending with an answer to the main research question. We will then discuss how our research fits into the broader picture of current artificial intelligence research. We end the chapter with directions for further research.

Subquestion 1 Which types of explanations are produced by knowledge bases?

We surveyed the literature on explanation for machine learning models and for Bayesian networks (chapter 2), although the focus lay on machine learning explanation. We found three machine learning explanation categorisations in the literature: Guidotti et al. (2018), Gilpin et al. (2018), and Lipton (2016). Guidotti et al.'s was the most extensive and provided a basis for our conceptualisation. They divide the field of machine learning explanation into 1) model explanation through a simplified proxy model, 2) outcome explanation by constructing a function that takes the input and returns an explanation for that particular input, and 3) graphical or textual model inspection. We found the other two categorisations to be subsumed by Guidotti et al.'s categorisation. The two methods on Bayesian network explanation that we covered in section 2.2.2 (Lacave et al. (2007) and Timmer et al. (2017)) also fit into the categorisation of Guidotti et al., showing that their categorisation is applicable to other black boxes than just machine learning models.

Subquestion 2 Which types of explanations are useful according to philosophy and social theory?

From both philosophy and social theory contrastive explanation emerged as a very useful explanation. For philosophy it is an operationalisation of causation (Strevens 2006). In social theory contrastive explanation is a means to select a relevant explanation from a host of explanations (Miller 2019). However, contrastive explanation has not yet been used in a major explanation algorithm, making it unfeasible to disregard other types of explanation in favour of contrastive explanation.

Subquestion 3.1 What are the requirements we elicit from the literature?

Subquestion 3.2 How can we modularise the ontology?

Subquestion 3.3 Which competency questions characterise the information of the ontology?

We required the ontology to capture both the types of explanations we elicited from the literature, as well as the internal structure the explanations exhibit. We identified five types of internal structure: 1) tree-based, 2) rule-based, 3) a bag of items, 4) graphical, and 5) textual. From the literature we also identified several types of knowledge bases. On a high level we observed machine learning models, Bayesian networks, and expert systems. On a more detailed level we observed various subtypes of machine learning models. The final requirement was that the ontology should capture a treelike structure to model a combined explanation consisting of multiple explanation instances.

From the requirements we drew three ontology modules: 1) the knowledge base producing the explanation, 2) the explanation itself, and 3) the hierarchy of explanations. The proof that the required information can indeed be extracted from the ontology was given by the answers to a set of competency questions (chapter 8). From each explanation tree and each explanation individual we want to extract the fact being explained, the type and the contents of the actual explanation, and the conceptualising knowledge bases.

Subquestion 3 What is the appropriate reference ontology?

We divided the explanation into an *explanandum* (the fact that is being explained) and an *explanans* (the facts that perform the actual explanation). The types of machine learning explanation categorised in the domain survey became subtypes of the explanans, including a detailed conceptualisation of the explanation structure. For example, some explanations are structured as a decision tree, while others consist of a set of rules. The explanandum explains a fact, which is a triple in the context of ontologies.

Two singular explanations are combined into an explanation tree by unification of the explanans of one with the explanandum of the other. After all, the fact that performs the explanation may itself be subject to another explanation. The intermediate fact then plays the role of both explanandum and (part of the) explanans. This allows us to define a chain of relations that constitutes a child relation between two explanations.

Subquestion 4 How should we implement the reference ontology to build the operational ontology?

We implemented the Description Logic formalisation in the Web Ontology Language, which required us to weaken the axioms to keep the ontology decidable. A chain of relations was not allowed in a subrelation of a transitive relation. So, the operational ontology is slightly less strict than the reference ontology.

Subquestion 5 How do we incorporate the explanation ontology into the Plasido engine?

Subquestion 5.1 How do we combine several instances of the ontology from various sources to construct an

We expanded Plasido with explanation functionality to create Explainable Plasido (chapter 9). This step functions as the proof of concept, showing that our ontology works in practice, and fulfils the practical goal we set ourselves in the introduction. The major addition to Plasido is an explanation combination algorithm. It takes as its input a trace of Plasido indicating how it derived a particular outcome. The trace is turned into an instance of the explanation ontology by querying the appropriate external knowledge bases.

This allows us to answer the main research question:

Main question How do we leverage ontologies to integrate the various explanations of a federated knowledge environment?

The ontology functions as a standard for communication. It is not merely a detailed taxonomy or rough categorisation of explanations, but the OWL implementation defines the exact format that corresponds to a particular explanation instance. The ontology is strict enough for the explanation combination algorithm to operate on any instance of the ontology. Also, the descriptive rather than prescriptive nature of ontologies makes sure that our ontology definition is flexible enough to sufficiently capture new types of explanation that are to be developed.

10.1 Discussion

Our research used the somewhat older technique of ontologies to provide an interoperability solution for the more recent developments in explainable data science. The field is usually called *Explainable AI* (XAI), but its techniques only consider explainable machine learning, because more and more data science applications keep getting developed, many of which are used in additional sensitive domains like monitoring teenagers or deciding whether kids go to the next grade. Our work, on the other hand, allows the possible inclusion of other AI techniques based on logic or probability.

Our approach currently does not seem to function for interactive explanations, even though this type may seem to be a promising way of explanation. After all, it can adapt itself in real time to the specific needs of the subject. A possible solution may be that we do not provide the explanation itself, but rather a link with arguments to load the actual explanation. Via the link the user can access a web page or application where the interactive functionality is present. However, a solution has to be found to include facts upon which that explanation rests. When linking to an external source that is not in the ontology format, we lose the ability to query the ontology.

10.2 Future research

Our first recommendation is to construct a graphical interface on top of the ontology. The concrete ontology is just as unreadable to professionals as to non-professionals, but the well-defined structure facilitates a mapping from ontology individual to graphical representation. This interface has to be grounded in psychological research on how people want an explanation administered to them (Miller 2019). Translating the ontology into a visual tree should pose few problems given the SPARQL-implementations of our

competency questions and the strict implementation of the ontology. This step is also very useful if Explainable Plaido is to be used in future TNO-projects.

Secondly, we propose to extend the ontology with classes to capture types of explanations not yet captured. The open-world nature of ontologies explicitly allows adding additional concepts that inherit from existing concepts. Another researcher can easily extend our Explanans-class with a new subclass, because our ontology is located at an online TNO-repository. Our ontology for the various types of knowledge bases can be fleshed out in much the same way. Explanation for case-based reasoning may seem an interesting type of knowledge base to capture, since its structure may overlap with types of machine learning explanation that use examples (Leake and McSherry 2005).

The last, and probably most interesting, line of research we would like to mention is the PAL-project (Neerincx et al. 2016), where ontologies are used to provide children with diabetes, as well as their parents and their doctors with an explanation of why the child should at a particular moment inject insulin. Our ontology can be extended with classes that define which explanations are satisfactory for children, which for their parents, and which for medical professionals. We already gave the structure of an explanation a detailed conceptualisation. It would for example take little effort to define a maximum number of rules a rule-based proxy may consist of for it to be considered a satisfactory explanation for children. Similarly, we can define the maximum depth of a tree-based proxy that is presented to the patient. The capabilities of the ontology reasoner allow the automatic inference of whether an explanation is judged to be interpretable for the patient, the parents, or the medical professionals.

Bibliography

- Adhikari, Ajaya (2018). “Example and Feature Importance-based Explanations for Black-box Machine Learning Models”. Master’s thesis. TNO/TU Delft.
- Angwin, Julia, Jeff Larson, Surya Mattu, and Lauren Kirchner (2016). “Machine bias”. *ProPublica*. URL: <https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing> (visited on 05/09/2018).
- Baader, Franz, Ian Horrocks, Carsen Lutz, and Uli Sattler (2017). *An Introduction to Description Logic*. Cambridge University Press.
- Biran, Or and Courtenay Cotton (2017). “Explanation and justification in machine learning: a survey”. In: *International Joint Conference on AI. Workshop on Explainable AI*, pp. 8–13.
- Brewster, Christopher, Harith Alani, Srinandan Dasmahapatra, and Yorick Wilks (2004). “Data driven ontology evaluation”. In: *Proceedings of the Fourth International Conference on Language Resources and Evaluation*. Ed. by Maria Teresa Lino, Maria Francisca Xavier, Fátima Ferreira, Rute Costa, and Raquel Silva. ELRA.
- Carral, David, Pascal Hitzler, Hilmar Lapp, and Sebastian Rudolph (2017). “On the ontological modeling of trees”. *arXiv:1710.05096*.
- DARPA (2016). *Explainable Artificial Intelligence Program*. Tech. rep. <http://www.darpa.mil/program/explainable-artificial-intelligence>: DARPA.
- Della Valle, Emanuele and Stefano Ceri (2011). “Querying the semantic web: SPARQL”. In: *Handbook of Semantic Web Technologies*. Ed. by John Domingue, Dieter Fensel, and James A. Hendler. Springer Science & Business Media, pp. 299–364.
- Doshi-Velez, Finale and Been Kim (2017). “Towards a rigorous science of interpretable machine learning”. *arXiv:1702.08608v2*.
- Falbo, Ricardo de Almeida (2014). “SABiO: systematic approach for building ontologies”. In: *Proceedings of the 1st Joint Workshop ONTO.COM/ODISE on Ontologies in Conceptual Modeling and Information Systems Engineering, co-located with the 8th International Conference on Formal Ontology in Information Systems (FOIS)*. CEUR.
- Gangemi, Aldo and Valentina Presutti (2009). “Ontology design patterns”. In: *Handbook on Ontologies*. Springer, pp. 221–243.
- Gilpin, Leilani H., David Bau, Ben Z. Yan, Ayesha Bajwa, Michael Specter, and Lalana Kagal (2018). “Explaining explanations: an approach to evaluating interpretability of machine learning”. *arXiv:1806.00069v2*.
- Glimm, Birte, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang (2014). “Hermit: an OWL 2 reasoner”. *Journal of Automated Reasoning* 53 (3), pp. 245–269.

- Gruber, Thomas R. (1993). “Toward principles for the design of ontologies used for knowledge sharing”. *International Journal Human-Computer Studies* 43, pp. 901–928.
- Guidotti, Riccardo, Anna Monreale, Franco Turini, Dino Pedreschi, and Fosca Giannotti (2018). “A survey of methods for explaining black box models”. *arXiv:1802.01933*.
- Hilton, Denis J. (1990). “Conversational processes and causal explanation”. *Psychological Bulletin* 107 (1), pp. 65–81.
- Horridge, Matthew and Mark Musen (2016). “Snap-SPARQL: a Java framework for working with SPARQL and OWL”. In: *Ontology Engineering*. Ed. by Valentina Tamma, Mauro Dragoni, Rafel Gonçalves, and Agnieszka Ławrynowicz. Springer, pp. 154–165.
- Kvamme, Håvard, Nikolai Sellereite, Kjersti Aas, and Steffen Sjursen (2018). “Predicting mortgage default using convolutional neural networks”. *Expert Systems with Applications* 102 (15), pp. 207–217.
- Lacave, Carmen and Francisco J. Díez (2002). “A review of explanation methods for Bayesian networks”. *The Knowledge Engineering Review* 17 (2), pp. 107–127.
- (2004). “A review of explanation methods for heuristic expert systems”. *The Knowledge Engineering Review* 19 (2), pp. 133–146.
- Lacave, Carmen, Manuel Luque, and Francisco J. Díez (2007). “Explanation of Bayesian networks and influence diagrams in Elvira”. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 37 (4), pp. 952–965.
- Leake, David and David McSherry (2005). “Introduction to the special issue on explanation in case-based reasoning”. *Artificial Intelligence Review* 24 (2), pp. 99–102.
- Lipton, Peter (1990). “Contrastive explanation”. In: *Explanation and its Limits*. Royal Institute of Philosophy Supplement. Ed. by Dudley Knowles. Vol. 27, pp. 247–266.
- (1991). *Inference to the Best Explanation*. Routledge, London.
- Lipton, Zachary C. (2016). “The mythos of model interpretability”. *arXiv:1606.03490*.
- Lohmann, Steffen, Stefan Negru, Florian Haag, and Thomas Ertl (2016). “Visualizing ontologies with VOWL”. *Semantic Web* 7 (4), pp. 399–419.
- McGuinness, Deborah L. and Frank van Harmelen (2004). “OWL web ontology language overview”. *W3C Recommendation*. URL: <https://www.w3.org/TR/owl-features/>.
- Miller, Tim (2018). “Contrastive explanation: a structural-model approach”. *arXiv:1811.03163v1*.
- (2019). “Explanation in artificial intelligence: insights from the social sciences”. *Artificial Intelligence* 267, pp. 1–38.
- Miller, Tim, Piers Howe, and Liz Sonenberg (2017). “Explainable AI: beware of inmates running the asylum. Or: how I learnt to stop worrying and love the social and behavioural sciences”. In: *IJCAI-17 Workshop on Explainable AI*, pp. 36–42.
- Motik, Boris, Peter F. Patel-Schneider, and Bijan Parsia (2012). “OWL 2 web ontology language. Structural specification and functional-style syntax (second edition)”. *W3C Recommendation*. URL: <https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/>.
- Musen, Mark A. and Protégé Team (2015). “The Protégé project: a look back and a look forward”. *AI Matters* 1 (4), pp. 4–12.
- Neerinx, Mark A., Frank Kaptein, Michael van Bekkum, Hans-Ulrich Krieger, Bernd Kiefer, Rifca Peters, Joost Broekens, Yiannis Demiris, and Maya Sapelli (2016).

- “Ontologies for social, cognitive and affective agent-based support of child’s diabetes self-management”. In: *Artificial Intelligence for Diabetes. 1st ECAI Workshop on Artificial Intelligence for Diabetes at the 22nd European Conference on Artificial Intelligence (ECAI 2016)*, p. 35.
- Neerinx, Mark A., Jasper van der Waa, Frank Kaptein, and Jurriaan van Diggelen (2018). “Using perceptual and cognitive explanations for enhanced human-agent team performance”. In: *Engineering Psychology and Cognitive Ergonomics*. Ed. by Don Harris. Cham: Springer International Publishing, pp. 204–214.
- Nouwt, Barry (2016). *Plasido Architecture*. Tech. rep. TNO.
- Nouwt, Barry and Jack Verhoosel (2018). “Integrating heterogeneous data sources using rule-based reasoning, backward-chaining and custom built-ins”. In: *Proceedings of the Posters and Demos Track of the 14th International Conference on Semantic Systems - SEMANTiCS2018* (Vienna, Austria). CEUR Workshop Proceedings.
- Obrst, Leo, Werner Ceusters, Inderjeet Mani, Steve Ray, and Barry Smith (2007). “The evaluation of ontologies. Toward improved semantic interoperability”. In: *Semantic Web. Revolutionizing Knowledge Discovery in the Life Sciences*. Springer, New York, pp. 139–158.
- Poveda-Villalón, María, Asunción Gómez-Pérez, and Mari Carmen Suárez-Figueroa (2014). “OOPS! (OntOlogy Pitfall Scanner!): an on-line tool for ontology evaluation”. *International Journal on Semantic Web and Information Systems* 10 (2), pp. 7–34.
- Presutti, Valentina and Aldo Gangemi (2016). “Dolce+D&S Ultralite and its main ontology design patterns”. In: *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*. IOS Press, pp. 81–104.
- Ribeiro, Marco Tulio, Sameer Singh, and Carlos Guestrin (2016). “Why should I trust you? Explaining the predictions of any classifier”. In: *Proceedings of the 22nd ACM Special Interest Group on Knowledge Discovery in Data (SIGKDD) International Conference on Knowledge Discovery and Data Mining*. ACM, pp. 1135–1144.
- Shearer, Rob, Boris Motik, and Ian Horrocks (Oct. 2008). “Hermit: a highly-efficient OWL reasoner”. In: *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2008 EU)*. Ed. by Alan Ruttenberg, Ulrike Sattler, and Cathy Dolbear. Karlsruhe, Germany.
- Strevens, Michael (2006). “Scientific explanation”. In: *Encyclopedia of Philosophy*. Ed. by D. M. Borchert. 2nd ed. Macmillan Reference, Detroit.
- Su, Xiaomeng, Mihail Matskin, and Jinghai Rao (2003). “Implementing explanation ontology for agent system”. In: *IEEE/WIC International Conference on Web Intelligence*, pp. 330–336.
- Tiddi, Iliaria, Mathieu d’Aquin, and Enrico Motta (2015). “An ontology design pattern to define explanations”. In: *Proceedings of the 8th International Conference on Knowledge Capture*. Association for Computing Machinery.
- Timmer, Sjoerd T., John-Jules Ch. Meyer, Henry Prakken, Silja Renooij, and Bart Verheij (2017). “A two-phase method for extracting explanatory arguments from Bayesian networks”. *International Journal of Approximate Reasoning* 80, pp. 475–494.
- Vrandečić, Denny (2009). “Ontology evaluation”. In: *Handbook on Ontologies*. Ed. by Steffen Staab and Rudi Studer. 2nd ed. Springer, pp. 293–313.

- Waa, Jasper van der, Marcel Robeer, Jurriaan van Diggelen, Matthieu Brinkhuis, and Mark Neerincx (2018). “Contrastive explanation with local foil trees”. *arXiv:1806.07470v1*.
- Whitehead, Alfred North (1938). *Modes of Thought*. MacMillan, New York.
- Xiao, Tianjun, Yichong Xu, Kuiyuan Yang, Jiaying Zhang, Yuxin Peng, and Zheng Zhang (2015). “The application of two-level attention models in deep convolutional neural network for fine-grained image classification”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 842–850.

Appendix A

SPARQL queries

All SPARQL queries use the prefixes defined in figure 7.1.

```
1 SELECT ?explanation ?subject ?predicate ?object
2 WHERE
3   { ?explanation rdf:type          tno:Explanation ;
4     tno:hasExplanandum ?_explanandum .
5     ?_explanandum
6       rdf:subject          ?subject ;
7       rdf:predicate        ?predicate ;
8       rdf:object           ?object
9   }
```

Listing A.1: What fact is being explained?

```
1 SELECT DISTINCT ?explanation ?type
2 WHERE
3   { ?explanation rdf:type          tno:Explanation ;
4     tno:hasExplanans ?_explanans .
5     ?_explanans rdf:type          ?type .
6     ?type rdfs:subClassOf tno:Explanans
7   }
```

Listing A.2: What is the function of the explanation?

```
1 SELECT ?explanation ?part
2 WHERE
3   { ?explanation rdf:type          tno:Explanation ;
4     tno:hasExplanans ?_explanans .
5     ?_explanans tno:hasExplainablePart ?part
6   }
```

Listing A.3: Which items does the the explanation consist of?

```

1 SELECT DISTINCT ?explanation ?knowledgeBase
2 WHERE
3   { ?explanation rdf:type          tno:Explanation ;
4     tno:isConceptualizedBy ?knowledgeBase
5   }

```

Listing A.4: Which KB is the explanation derived from?

```

1 SELECT DISTINCT ?knowledgeBase ?type
2 WHERE
3   { ?knowledgeBase
4     rdf:type          tno:ExplainableKnowledgeBase ;
5     rdf:type          ?type ;
6     dul:conceptualizes ?_explanation .
7     ?_explanation
8     rdf:type          tno:Explanation
9     VALUES ?type { tno:BayesianNetworkKnowledgeBase
10                   tno:ExpertSystemKnowledgeBase
11                   tno:MachineLearningKnowledgeBase }
12   }

```

Listing A.5: Which type among expert system, ML model and Bayesian network does the KB belong to?

```

1 SELECT DISTINCT ?knowledgeBase ?type
2 WHERE
3   { ?knowledgeBase
4     rdf:type          tno:ExplainableKnowledgeBase ;
5     rdf:type          ?type ;
6     dul:conceptualizes ?_explanation .
7     ?_explanation
8     rdf:type          tno:Explanation
9     FILTER ( ( ( ?type != owl:NamedIndividual ) &&
10              ( ?type != dul:Agent ) ) &&
11             ( ?type != tree:TreeNode ) )
12   }

```

Listing A.6: Which combination of subtypes identifies the KB?

```

1 CONSTRUCT
2   {
3     ?node rdf:type tree:TreeNode .
4     ?node tree:hasChild ?child .
5     ?node tree:hasOutDegree ?outDegree .
6     ?node rdfs:label ?label .
7     ?node rdf:type owl:NamedIndividual .
8     ?root rdf:type tree:RootNode .
9     ?leaf rdf:type tree:LeafNode .
10  }
11 WHERE
12   { ?_explanation
13     rdf:type tno:Explanation ;
14     tno:hasExplanandum ?node ;
15     tree:hasOutDegree ?outDegree
16   OPTIONAL
17     { ?_explanation
18       tno:hasExplanationChild ?_explanationChild .
19       ?_explanationChild
20         tno:hasExplanandum ?child
21     }
22   OPTIONAL
23     { ?_explanation
24       rdf:type tree:RootNode ;
25       tno:hasExplanandum ?root
26     }
27   OPTIONAL
28     { ?_explanation
29       rdf:type tree:LeafNode ;
30       tno:hasExplanandum ?leaf
31     }
32   }

```

Listing A.7: Construct a simplified fact tree.

```

1 CONSTRUCT
2   {
3     ?node rdf:type tree:TreeNode .
4     ?node rdf:type owl:NamedIndividual .
5     ?node tno:isConceptualizedBy ?kb .
6     ?node tree:hasOutDegree ?outDegree .
7     ?node tree:hasChild ?child .
8     ?leaf rdf:type tree:LeafNode .
9     ?root rdf:type tree:RootNode .
10  }
11 WHERE
12   { ?node   rdf:type          tno:Explanation ;
13     tno:isConceptualizedBy ?kb ;
14     tree:hasOutDegree      ?outDegree
15   OPTIONAL
16     { ?node   tno:hasExplanationChild ?child }
17   OPTIONAL
18     { ?leaf   rdf:type   tno:Explanation ;
19       rdf:type   tree:LeafNode
20     }
21   OPTIONAL
22     { ?root   rdf:type   tno:Explanation ;
23       rdf:type   tree:RootNode
24     }
25   }

```

Listing A.8: Construct a simplified knowledge base tree.

```

1  CONSTRUCT
2  {
3      ?explanation rdf:type tno:Explanation .
4      ?explanation rdf:type tno:ExplanationTreeNode .
5      ?explanation rdf:type owl:NamedIndividual .
6      ?explanation tree:hasOutDegree ?outDegree .
7      ?explanation tno:hasExplanans ?explanans .
8      ?explanation tno:hasExplanandum ?explanandum .
9      ?explanation tno:hasExplanationChild ?descendant .
10     ?leaf rdf:type tree:LeafNode .
11     ?root rdf:type tree:RootNode .
12 }
13 WHERE
14 { ?explanation rdf:type          tno:Explanation ;
15   tno:hasExplanandum ?explanandum ;
16   tno:hasExplanans ?explanans ;
17   tno:isConceptualizedBy ?kb .
18   ?kb rdf:type ?kbType
19   VALUES ?kbType { tno:BayesianNetworkKnowledgeBase tno:MachineLearningKnowledgeBase }
20   OPTIONAL
21   { ?explanation tree:hasDescendant ?descendant
22     FILTER EXISTS { { ?kbPrime dul:conceptualizes ?descendant ;
23                       rdf:type          tno:MachineLearningKnowledgeBase
24                     }
25                     UNION
26                     { ?kbPrime dul:conceptualizes ?descendant ;
27                       rdf:type          tno:BayesianNetworkKnowledgeBase
28                     }
29                     FILTER NOT EXISTS { ?middle rdf:type          tno:Explanation ;
30                                           tno:isConceptualizedBy ?kbPrimePrime
31                                           { ?kbPrimePrime
32                                             dul:conceptualizes ?middle ;
33                                             rdf:type          tno:BayesianNetworkKnowledgeBase
34                                           }
35                                           UNION
36                                           { ?kbPrimePrime
37                                             dul:conceptualizes ?middle ;
38                                             rdf:type          tno:MachineLearningKnowledgeBase
39                                           }
40                                           ?middle tree:hasDescendant ?descendant ;
41                                           tree:hasAncestor ?explanation
42                                         }
43                   }
44   }
45   OPTIONAL
46   { ?leaf rdf:type          tno:Explanation ;
47     tno:hasExplanandum ?explanandum ;
48     tno:hasExplanans ?explanans ;
49     tno:isConceptualizedBy ?kb .
50     ?kb rdf:type ?kbType
51     VALUES ?kbType { tno:BayesianNetworkKnowledgeBase tno:MachineLearningKnowledgeBase }
52     FILTER NOT EXISTS { ?treeDescendant
53                       tree:hasAncestor ?leaf ;
54                       tno:isConceptualizedBy ?kbPrimePrimePrime
55                       { ?kbPrimePrimePrime
56                         dul:conceptualizes ?middle ;
57                         rdf:type          tno:BayesianNetworkKnowledgeBase
58                       }
59                       UNION
60                       { ?kbPrimePrimePrime
61                         dul:conceptualizes ?middle ;
62                         rdf:type          tno:MachineLearningKnowledgeBase
63                       }
64                     }
65   }
66   OPTIONAL
67   { ?root rdf:type          tno:Explanation ;
68     tno:hasExplanandum ?explanandum ;
69     tno:hasExplanans ?explanans ;
70     tno:isConceptualizedBy ?kb .
71     ?kb rdf:type ?kbType
72     VALUES ?kbType { tno:BayesianNetworkKnowledgeBase tno:MachineLearningKnowledgeBase }
73     FILTER NOT EXISTS { ?treeAncestor
74                       tree:hasDescendant ?root ;
75                       tno:isConceptualizedBy ?kbPrimePrimePrimePrime
76                       { ?kbPrimePrimePrimePrime
77                         dul:conceptualizes ?middle ;
78                         rdf:type          tno:BayesianNetworkKnowledgeBase
79                       }
80                       UNION
81                       { ?kbPrimePrimePrimePrime
82                         dul:conceptualizes ?middle ;
83                         rdf:type          tno:MachineLearningKnowledgeBase
84                       }
85                     }
86   }
87 }

```

Listing A.9: Construct a simplified tree hierarchy.