Deploying the Semantic Web in the Welfare Environment

A DUTCH CASE STUDY

Master Thesis Computer Science

RADBOUD UNIVERSITY NIJMEGEN

Jos Claessens

August 25, 2009

Thesis nr: IC 612

Supervisors

Radboud University:	Prof. dr. ir. Th. (Theo) van der Weide
PPC:	Jan-Willem Schoenmakers
Second reader:	dr. P. (Patrick) van Bommel

in memory of Sjaak Claessens

"A #2 pencil and a dream can take you anywhere"
- - Joyce A. Myers

Preface

In the summer of 2008, I got in contact with Partners in Professional Computing (PPC). The main question we had after our meeting: "Does the Semantic Web have any benefits, and if so: how should we exploit them?" An entire world opened up for me when exploring this exciting area of web technology. The subjects at hand happened to fit my computer science interests nicely: web technology, conceptual modeling and logics, deployed in an environment closely related to human interests. I was pleasantly surprised with the clear human relevance radiating from the Social Map. My first thanks therefore go out to my colleagues at PPC, in particular my supervisor Jan-Willem Schoenmakers, for their assistance in my analysis of their daily environment.

As it happens, one of the most important aspects of academic writing is its academic character. Right from the start, I found an excellent supervisor in the person of Theo van der Weide. I dare to express here that I couldn't have finished this thesis without him. He made me realize that a pencil and a dream really can take you anywhere*. Unfortunately, the associated quote belongs to Joyce A. Myers. Nevertheless, I thank Theo very much for showing me how to deal with all kinds of uncertainty that are typical for the process of becoming a master.

I would like to thank many people for their seemingly eternal patience and tranquility, especially when I sometimes lost mine. In the first place, this goes for those closest to me: my Father, parents, family and of course Laura – you are wonderful. Thanks to all friends that kept motivating me with comforting words when it was needed – or when it was not needed, for that matter.

Finally, I would like to thank you, reader, for taking the effort to read at least part of these roughly 28,000 words describing the results of my research. After all, I wrote them for you. I hope you enjoy reading them at least as much as I enjoyed writing them.

Jos Claessens

Nijmegen, August 2009

* or 23 figures, at least

Abstract

In this thesis, we investigate the abstraction gap between the real world and knowledge-intensive applications supporting tasks. We propose to bridge this gap by constructing a machine-interpretable conceptual model. To illustrate our method, we perform a case study in the welfare environment, deriving a model from an existing Dutch web-based database-driven information system. Our model is constructed using the established languages PSM and ORC and subsequently interpreted using a novel logics language called PSL. The result is suitable to support any novel web application in the welfare domain, using Semantic Web techniques. We investigate how these techniques can be used to enhance efficiency and effectiveness of the resulting application, compared to the current approach.

Table of Contents

1 Introduction

We have grown used to the World-Wide Web (WWW), which was originally designed in the early '90s by Tim Berners-Lee [1]. The WWW can be characterized as a collection of linked *documents*. The innovative part of the original web was the use of a uniform format to mark up web documents and - more importantly - using hyperlinks to link different documents and other resources to each other. Since its introduction, the Web has grown explosively. Currently, automatic spiders are constantly wandering around to determine its structure, which is ever changing. The Web being so large, human users try to find what they want using powerful search engines. However, this process still often proves difficult due to ambiguity of search terms or the sheer size of possible matches. It's up to the user to further refine his search query and formulate it in a way that's probably successful, given the search engine's rather limited possibilities.

The most important reason wide-spread search engines currently feel limited is the markup of the documents they have to work with. Most of the structural elements of (X)HTML documents are aimed at presentation, and therefore, *human* understanding. *Automatic* reasoning about content remains difficult. Annotating this content with semantical metadata is a major step forward: it enables more intelligent processing of user queries and related documents, comparing results not only on the textual level but also on the level of their meaning. By using standardized formats for representing knowledge and reasoning rules, the Web is changing into a Semantic Web. Semantic documents may be processed automatically, answering information need by using underlying data concepts, rather than only the used representation. Intelligent searching and reasoning will become much easier and consequently more advanced.

The Semantic Web's effect will be most notable in an environment where users have a complex or (textually) ambiguous information need that cannot easily be converted to "Googlian" search terms. In this thesis, we focus on one such environment, which we call the "welfare environment": everything related to residents with a specific welfare need, for example impaired or chronically diseased. Such residents are also called "clients", a label inspired by their common need for help to find their way in regular society. Special service organizations have emerged to assist conventional health care in this task, which lies beyond their responsibility. A (web-based) information system can support clients by answering their specific information need. We call such information system a Social Map. Clients may query the system directly or visit a service organization, where a counselor may use it to support his own knowledge of the welfare environment. Improvements upon the Social Map reflect upon the quality of counseling sessions.

There are three major service organizations operating in the Dutch welfare environment: MEE[1], GGD[2] and the union of Dutch libraries (VOB)[3]. MEE is most specifically targeted at servicing people that 'need a hand', our target group. The GGD is

[1] http://www.mee.nl
[2] http://www.ggd.nl
[3] http://www.debibliotheken.nl

more generally targeted at extending health care 'outside hospital walls', while the VOB is most generally oriented, providing residents with all sorts of information, including information about their social welfare environment. Although the activity focus is different, all three have developed a Social Map system to suit their needs (and those of clients).

Our research has been done in cooperation with Partners in Professional Computing (PPC)[4], the technical partner of MEE. The BSK system, as their Social Map is called, strongly resembles the SoCard[5] system used by the GGD and the G!DS[6] system used by the VOB. In fact, a cooperation between all three parties has been established in April 2008 to start integrating the different information systems and make use of each other's knowledge. However, due to the fact that the Social Maps have been developed independently and with a different view on the environment in mind, this task proves very difficult. Even when extending a single Social Map, typically to resemble client's desires more closely, practical design issues occur more often than not.

In this thesis, we will investigate the Dutch welfare environment in a case study, to determine how to approach the deployment of Semantic Web technologies. We believe that successfully connecting to the emerging Semantic Web can significantly benefit service organizations to help clients and their counselors in their information need, both individually and cooperatively. By approaching the matter in a generic way, we aim to ensure that resulting recommendations should also be useable for comparable integration challenges in other environments.

The remainder of this chapter is organized as follows: in section 1.1 we will provide some background information about the current Social Map and its environment. In section 1.2, we will present the setup of our research towards a novel approach.

1.1 Background

We will first briefly investigate the development of web technology, to gain insight in the technical environment around the Social Map. Then, we will look more closely into the Social Map itself. We identify the key challenge in the current approach, which leads towards the main research question of this thesis.

1.1.1 Web technology

The inventor of the Web, Tim Berners-Lee, originally designed it as "an information space, with the goal that it should be useful not only for human-human communication, but also that machines would be able to participate and help" [2]. Interestingly, already in 1998, when the worldwide creation of web pages had led to machine-supported human-human information sharing, he identified certain key missing elements that would be needed to truly reach the full potential of the Web as originally intended: to enrich data so that machines can process and reason about them. However, realistically

[4] http://www.ppcnet.nl
[5] http://www.socard.nl
[6] http://gids.bibliotheek.nl

enough, he admitted those elements were technically out of reach at that moment. In the years that followed, the potential of the Web targeted at humans advanced dramatically. Reflecting upon these changes, we can identify some developmental stages.

Firstly, the number of web pages and links between them increased as the benefits of "being online" became clear for both suppliers and consumers of information. The development of search engines further changed the way the Web was used; nowadays humans can relatively easily find what they want using appropriate queries. We can summarize this first major development stage as follows: it was made possible to publish and access information designed for human consumption.

The next major developmental stage is commonly referred to as Web 2.0. It roughly describes the Web as we experience it today. Ankolekar et al. identified three main features introduced in this "new Web": *community, mashups* and *AJAX* [3]. Just like search engines, these features were gradually introduced in the existing web and afterwards considered fundamental enough to be worthy of a special designation. The *community* feature concerns the ability of regular web users to collaborate and share information, creating a collection of information for all to use. This collection could not have been made by any individual user, for he or she would not know as many other users as there are contributors to a community web page. Furthermore, the permanent availability and automatic processing of a web site provides the community with a natural environment that replaces all organization and structuring effort that would be hard to reach with human effort. Because users gain more from the system than they are required to put into the system, hardly any external motivation is required and the emerging of a community mostly depends on its familiarity in the web environment. The *mashup* feature is about pulling together different sources of information, to create a new source that is greater than the sum of its parts. Examples include a personal homepage, where parts of different other pages are shown together for easy overview, but also the integration of "things with a location" with Google Maps[7], to show their location in a much more intuitive way (namely on a map). Examples of this include housing overviews, online market places, but also just an easy map view of just a company's location. The mashing up of different information sources already stimulates web application developers to provide interfaces for other web developers, so cooperative efforts may emerge which benefit all involved parties. Finally, AJAX is the technological achievement that largely made the other two features possible. The main benefit provided by AJAX is *asynchronous communication*. This allows more intuitive and advanced user interfaces. Also, the responsiveness of mashed up web pages increases when different parts may be retrieved and refreshed independently.

Web 2.0 provides humans with a solid platform in which human-human communication is possible. However, this is not reflected in human-machine or machine-machine communication. All efforts related to interpretation, combining and evaluation of information are left to the human user. This state of affairs calls for the next major development, which was already called the Semantic Web in the 1998

[7] http://maps.google.com

visionary document by Tim Berners-Lee [2]. As outlined in a recent evaluation by Jorge Cardoso [4], this vision is rapidly becoming reality. The key idea of the Semantic Web vision is to provide a machine-readable representation of knowledge, so that facts about concepts and relations are accessible independent of the (human-targeted) representation. Using logics, machines can efficiently reason about world knowledge and answer questions more accurately. Users may search the Web "as though it were one giant database, rather than one giant book" [2]. The abstraction from representation will not only help machines to "understand" the information that is stored, it will also help integrate different information sources or reconcile different views upon the same world. By explicitly stating how all views are related to each other, we may ultimately reach a total model of all views on all world concepts, which represents all knowledge known and published. In this thesis, we will humbly start by looking at a small part of this global environment: the Dutch welfare environment.

1.1.2 The Social Map

The purpose of the Social Map is to provide Dutch residents with information about their welfare environment. It is aimed at residents with a specific welfare need, such as chronically diseased or impaired, which we call *clients*. But anyone might benefit from the provided information. We can think of the Social Map as a conceptual application, which tries to cover all information that clients in the welfare environment are interested in. Let's take a look at the welfare environment's environment.

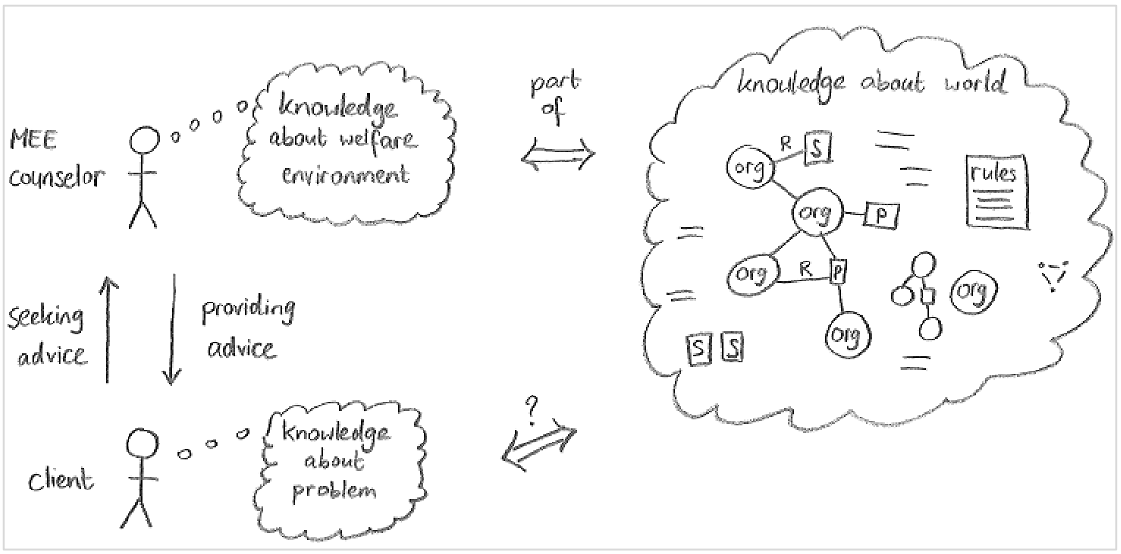

Figure 1 – The informal welfare environment

In Figure 1, we see the counselor and client depicted, including the interaction between them. The associated thought clouds contain their main contribution to this interaction. The client has some problem, which is obviously in some way related to the real world. But typically, once a client fulfils the role depicted here, this connection is unclear for him. Therefore, he seeks advice from a counselor whose knowledge is more focused upon the welfare environment. Just like the welfare environment itself, this knowledge is part of the knowledge about the whole world. Of course, we cannot place all world knowledge in just a few square centimeters. But we tried to give an idea of what we mean: related organizations (org), offering products (P) and services (S), sometimes only when certain rules (R) are met. All these may play any role, for we may view the real world as an environment in which everything is possible.

We may also look at the welfare environment at a formal level. By this, we mean all systems, information and technical structures *about* the (welfare) environment. This is depicted in Figure 2 below.

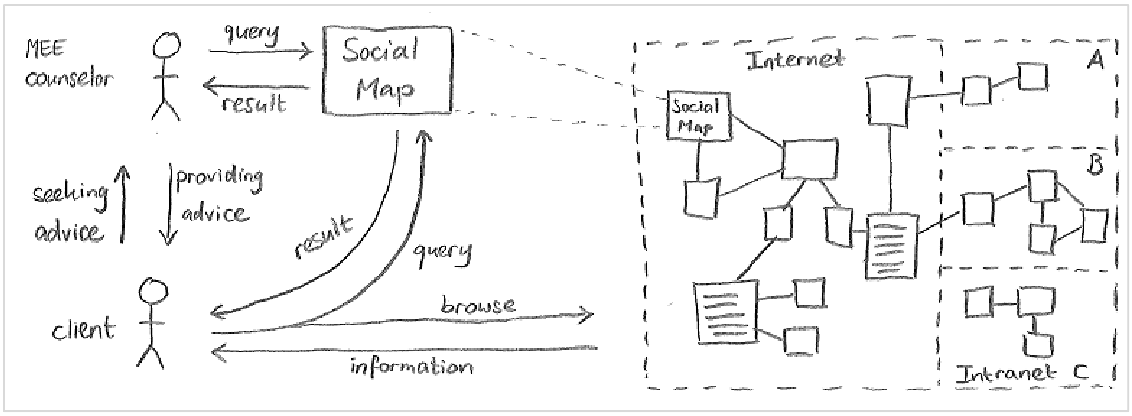

Figure 2 – The formal welfare environment

There is a clear resemblance between the formal and informal environment. Note that the client and counselor, as well as their interaction, are still informal in this picture. In fact, Figure 2 depicts the current way of working. Knowledge outside of system boundaries is omitted. What remains is the Social Map, which is a part of a larger network of applications and information, the internet. For knowledge sharing purposes, this may be regarded as the formal version of the real world, together with other networks which are more private (but may nevertheless contain useful information for some areas). Interaction with formal systems is not just a matter of thought invocation, but rather based on a query or browsing, gaining feedback which is hopefully useful. As we can see, the counselor and client may both retrieve information from the Social Map. The length of arrows illustrates a bit how difficult the process involved is.

This Social Map, representing this welfare environment, will be the center of our case study. It is a deviously simple single rectangle in Figure 2, but we will now have a closer look to appreciate encountered difficulties.

As we saw before, there are three major service organizations in the Netherlands: MEE, GGD and VOB. Each has a different activity focus and a corresponding view on how the welfare environment should be modeled. These views are not formally modeled, but they are the foundation of their different Social Map applications, which look like the abstract conceptual one in their own way. They have been implemented as a web application, enabling easy access and maintenance, as is a common way to design an information system nowadays. This situation has been depicted in Figure 3.

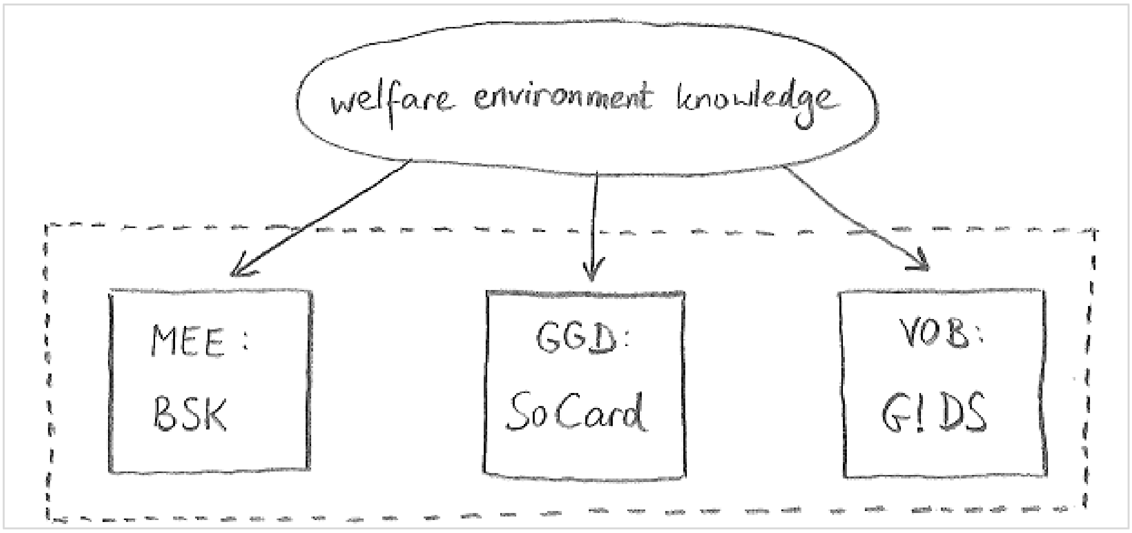

Figure 3 – Social Map approaches

In this figure, "welfare environment knowledge" is defined by all knowledge that could possibly be useful for clients. Thereby, we can neatly reference an ungraspable part of total world knowledge. We hope you appreciate the joys of abstract conceptual modeling. The different Social Map approaches each try to model this knowledge. Some observations about this situation are worth noting.

- There should be a certain conceptual similarity between the different Social Maps. All three parties could therefore benefit from each other's knowledge, while still respecting the different ways to make use of it.
- Knowledge from the welfare environment (such as information about available organizations, services or products) is possibly stored in three different ways, or not captured in all three systems. As a result, it costs more effort than desired or the scope is too limited.
- Clients or other web users have to "translate" their information retrieval process to match the different approaches and may as such miss out information or at least need to combine results from multiple sources.

As a result of these considerations, there is a desire to integrate the different approaches and share present knowledge. This has led to the cooperation of the three service organizations, trying to interconnect their data models. However, since there is no formal conceptual model of the welfare environment and the applications and the underlying data models have a different nature, this integration process quickly stifles and is restricted to sharing very common information about which all views agree, for example the name and address of an organization. Though it is possible to use this kind of basic information to extend an individual Social Map, it is unsatisfactory.

This lack of data structure interoperability is typical for a data-intensive web application. Note that this problem exists even in a thorough cooperation of technical partners; connecting to related third party applications will prove even harder. In this thesis, we are going to analyze why this problem occurs and propose an abstract, general approach towards dealing with it.

1.2 Research outline

In this section, we will provide an overview of our research. We will start with the identification of what we regard as the main problem encountered in contemporary application deployment. Based on this analysis, we will set out the path we follow during the rest of this thesis.

1.2.1 Problem identification

Deploying a data-intensive (web) application requires some sense of the environment about which data will be stored. The formalization and transformation process is far from simple, since the environment needs to be modeled towards an application-specific data structure. No matter how the resulting data structure might become, there will always be a gap between it and the way people in the environment work and think. Figure 4 illustrates various design steps which are typically implicitly or explicitly taken during data-intensive application development.

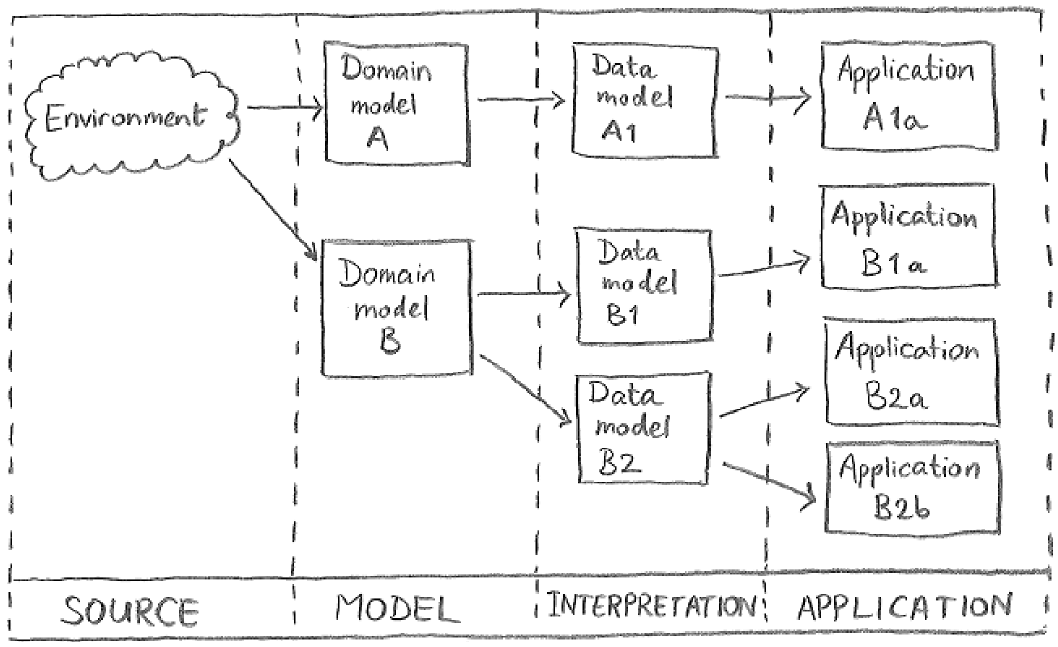

Figure 4 – Application abstraction layers

We distinguish four main abstraction layers: source, model, interpretation and application. The first layer is the environment itself. This is where typically the user base and real information, objects and relations reside. In our case, this is of course the welfare environment. An application which belongs in this environment has some connection with it. Note that since we are concerned with data-intensive applications, the connection typically has to do with data from the environment. Therefore, the development layers are targeted at deriving a suitable supporting data model which has its roots in the environment. Each design step, visualized by an arrow, introduces a set of decisions, errors and concessions. This is why two different models, though sharing the same direct root, are different. The further away this shared root is, the larger the difference between elements of the same layer becomes. We are interested in three main difficulty areas encountered in the current approach:

(1) satisfactorily interconnecting two applications
(2) extending an application
(3) satisfactorily interconnecting an application with the rest of the world

When we look at Figure 4, we can see why targets from these areas are so difficult.

Target 1 gets more difficult the more the roots of the involved applications differ from each other. For example, interconnecting application A1a with B2b will prove much more difficult than interconnecting B2a with B2b. A satisfactory connection can typically be made on the nearest commonly agreed model, establishing a translation between different specializations after that point. The nearest common model of A1a and B2b doesn't exist, it is the informal environment. B2a and B2b share the same interpretation model and may therefore relatively easily be interconnected.

Target 2 may prove difficult when there is no clear path between the source layer and the application layer, for the desired extension. Typically, an application extension comes forth from a desire in the real world environment (the source layer). This desire is typically not simply transformed into a comparable desire in the application layer.

Target 3 is perhaps the most ambitious of them all, trying to connect different applications based on a connection between different environments. Nevertheless, this is what users are trying to achieve when browsing the web using different solitary applications, which can be combined to achieve a larger goal.

As we see, even when all design steps are explicitly followed and elaborated, some frequently occurring targets are difficult to achieve. Additionally, domain model and data model layers are often unknown or imperfect, which makes connecting different information sources even more difficult. It would be a tremendous help if the environment were formally modeled independent of actual implementation approaches. This eliminates the non-formal common ancestor that is often the only ancestor known to all applications, creating a formal model that serves as a central interoperability network that relates to all specific implementations and by which all can use each other without needing to construct their own transformations.

To do this, the environment should be formally described in a standard way. The resulting model is about the *meaning* of concepts in the environment, in contrary to the *representation*. This separates the challenge of what the environment looks like from the challenge of how its data should be stored and accessed. Reaching such a semantic

structure, called an *ontology*, is one of the main goals of the Semantic Web. Creating an ontology is the matter of *ontology engineering* and therefore beyond our scope.
However, we may just assume that such ontology were in place, replacing the informal source called the welfare environment with a welfare ontology counterpart. Still, there should be a path between this new source and future applications. This is the second goal of the Semantic Web: real life applications that make use of the common general source model and extend it across all respective layers. In this utopic vision, all our three difficult targets are within reach, since everything is connected by a machine-interpretable semantic structure. Within this vision, we shall now identify the most relevant part which will be the topic of our research.

1.2.2 Research topic

Currently, a lot of research effort is spent towards ontology engineering and application-specific concerns around the Semantic Web. When we look at the Semantic Web goals, there is some sense in these topics: the first is targeted at creating an ontology, while the second is targeted at deploying applications within the new semantic structure. However, when we put it in the perspective of Figure 4, we see that most work is focused at the "beginning" (layer 1) and the "end" (layer 4). Interestingly, the missing link between both ends is rarely investigated in a structural and general manner. We believe such investigation is essential for successful deployment of the Semantic Web in any environment, so we will focus our research on the creation of a suitable domain model and interpretation model. This leads to our main question:

How should the abstraction gap between environment and application be bridged?

1.2.3 Thesis outline

Our research will have the form of a case study. By taking the MEE/PPC view upon the welfare environment for granted, we will construct a transformation design in two phases. The first phase will be targeted at the model layer. We will construct a domain model of the welfare environment in chapter 2 and 3. After that, we will have a literature overview intermezzo in chapter 4, investigating what the Semantic Web really is and what others have done towards reaching its goals. In chapter 5, we will extend our domain model with a general transformation method towards the interpretation layer. The combined result will largely fill up the gap found between the source and application layer. We conclude in chapter 6, with a reflection upon our case study, briefly summarizing its value for the general case of application development for the Semantic Web. Finally, we will suggest some topics for future work.

2 Modeling the Welfare Environment

In chapter 1, we saw the *domain model* placed in the second layer of application development (Figure 4). It has its roots in the actual (welfare) environment. In this chapter, we will prepare the construction of such a model. First, we will define a domain model more closely in section 2.1. Then, we will investigate the way such models are typically constructed in section 2.2. We will conclude this chapter in section 2.3, by shortly introducing the languages we will use to create our domain model.

2.1 *What is a domain model?*

The purpose of a model may be formulated as follows: "a model tries to grasp (describe) the essence of some part of the world around us" [5]. This definition refines our initial view with a small addition: the *essence*. Apparently, we are not simply trying to grasp the world or some part of it. Since a definition tends to become worse when the essence is removed, we choose to keep it. So what does this essence look like?

We are constructing an *administrative* model, the purpose of which is to construct a "shadow world (...) powerful enough to answer all kinds of questions about the state of affairs in the shadowed world" [5]. A typical information system has the same purpose. Common practice in information system design is to store only data that is really needed, to save storage space. In other words: only store *elementary facts*. The rationale behind this is that when (preferably all) essential elementary facts are known, we can easily derive more complex facts from them for occasional interests by using temporary computing power instead of using permanent storage.

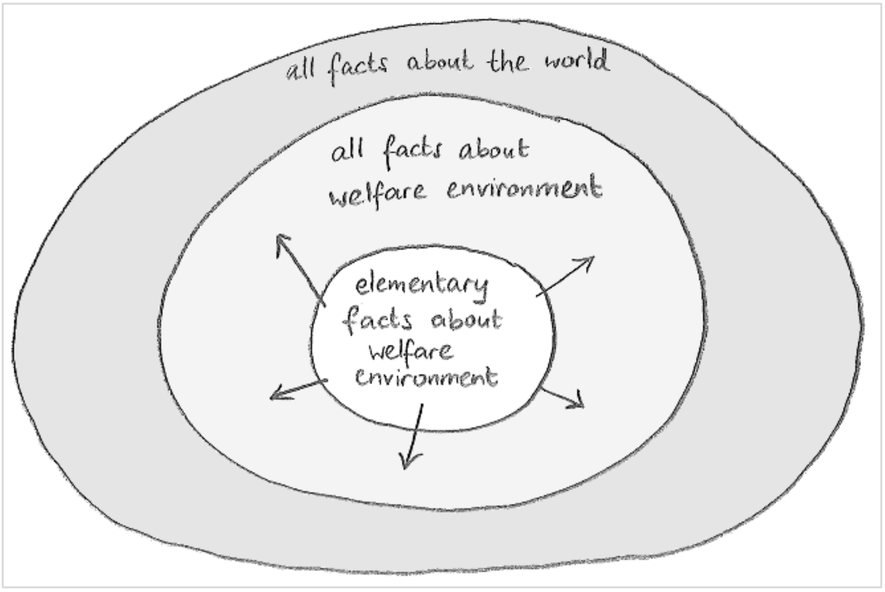
all facts about the world
all facts about welfare environment
elementary facts about welfare environment

Figure 5 – Fact hierarchy

This brings us to the essence of a domain model: (1) elementary facts from the shadowed world, and (2) a grammar for deriving more complex facts, which is maximally simple and maximally expressive. This essence is depicted in Figure 5. As we see, the core of our model is a base of elementary facts, depicted by the inner space. Furthermore, there are derivation rules, combining elementary facts to construct facts in the larger space that contains all facts about the welfare environment. These derived facts are visualized by arrows, accounting for their relation with elementary facts. Facts that are about the world outside of the welfare domain are neglected, but of course the welfare environment is not really isolated.
By using fact orientation, we focus on actual information, communication and facts from the welfare domain that describe the most essential things, as objectively as possible. By doing this, we create a model that is an abstract shadow version of the subjective environment as it is experienced by different people. Therefore, it is a suitable method to create a model in the second layer of application development. The shadow world is easier to grasp, maintain and reason about, while it is still widely applicable due to its abstract general purpose.

2.2 *Way of working*

Now that we know *what* we are trying to gather, we will look at *how* we may gather it. In this process, we can identify an interaction between two roles: the *domain expert* versus the *system analyst.* The domain expert role is much like the counselor role we already saw. He knows a lot about the target domain, and he can be asked to formulate sentences that express facts from this domain. Essentially, the domain expert may provide all possible facts, but lacks the bird-eye view needed to structurally sort them out. This bird-eye view is provided by the system analyst, who is capable of (formally) structuring facts and determining what to consider elementary and how to define the grammar to derive the remaining facts. This interaction is depicted in Figure 6, adapted from [5].

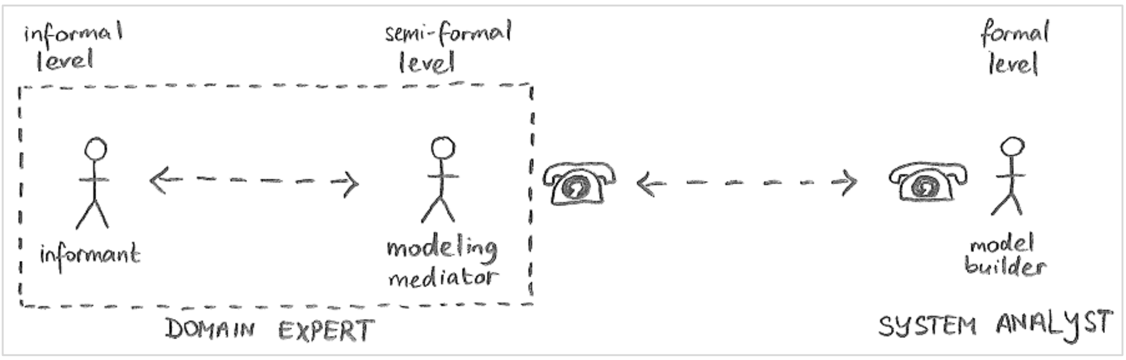

Figure 6 – Interaction to construct a model featuring domain expert and system analyst roles

We can see that the domain expert role actually consists of two different roles: the informant and the modeling mediator. How these roles are implemented in a concrete situation differs, but generally the informant role is least specifically targeted at one person. It consists of the knowledge, thoughts, processes and communication that takes place inside the target domain. The informant can be regarded as a spokesperson for the domain that knows a lot about these things. On the other hand, the modeling mediator forms expressions for the system analyst, based on his communication with the informant. These expressions may be seen as suggestions for facts that could be part of the constructed model. Whether they really will become part of the model is up to the system analyst. Where needed, he will communicate with the domain expert to clarify unclear or ambiguous fact sentences. Notice the telephone icon associated with this communication. It depicts the difference in nature between both communication lines: the modeling mediator communicates with the informant by informal language (as natural as it gets, including circumstantial influences like facial expressions or intonation) while he communicates with the system analyst using formal language. There are two main differences: formal language is restricted in form (only spoken text, hence the telephone symbol) and it is controlled (not all expression power of a regular natural language is suitable for a model). The communication between modeling mediator and system analyst is strictly fact-oriented. As we can see, the mediator role is crucial in that it provides the translation between the natural informal world and the structured formal world. The quality of the model depends on the quality of mediating fact sentences. To maximize the probability of success, assuming that the domain expert and system analyst have a fixed quality, we should focus on the *language* used to communicate. A well-chosen language is a key success factor, and therefore it deserves special attention.

2.3 *Our model languages*

The model language should meet two requirements. First of all, it should be *formal* so its meaning is clearly defined and strong reasoning can be applied. Secondly, it should be *understandable* for people in the welfare domain, who generally have less understanding of mathematical languages and concepts. These people use natural language, which tends to be ambiguous and informal. Meeting both requirements seems to be a challenge, but when you think of it, it is the very nature of computer science: trying to bridge the gap between computer and human in such a way that both worlds understand each other and that the bridge provides a desired connection between both. Therefore, it's not surprising that nowadays there are some very mature and widely used modeling languages that are both formal and understandable. They make use of something called *structured natural language*. The key idea of this is that one is allowed to use natural language to make expressions, but only when the expression meets certain conditions. These restrictions ensure that the expression is predictable enough to be interpretable in the formal domain, while allowing for maximized natural expressiveness. Resulting sentences are generally "almost natural"; they may just sometimes look a bit odd around the points where the structural restrictions were most hard to meet. We will make use of this kind of language to create our domain model. Recall that it will consist of elementary and derived facts.

The first language of choice is very similar to Object Role Modeling (ORM), first introduced by Terry Halpin [6]. ORM is widely used for conceptually modeling a domain based on natural language examples from the domain itself. The resulting model is a formal basis from which a desired implementation may be derived, which is typically an information system that contains data about the modeled domain. We will make use of an ORM variant called the *Predicator Set Model* (PSM), formalized by Ter Hofstede et al. [7]. Compared with traditional ORM, it contains some additional constructs which provide greater expressivity. We will summarize the main idea and used constructs in section 2.3.1. With PSM, we will build a model of the welfare environment, containing elementary facts and a framework for constructing derived facts.

The second language is Object Role Calculus (ORC). While ORM is targeted at structurally modeling the environment, ORC is targeted at defining additional derived facts and logical rules, as well as reasoning. A detailed overview of ORC is provided in the PSM introductory paper by Ter Hofstede et al. [7], where it is called LISA-D. It was conveniently designed as an extension upon PSM, so using both languages together for our domain model is quite straightforward. We provide a summarized version in section 2.3.2.

2.3.1 Predicator Set Model

With the Predicator Set Model, we can define an information structure using mainly the following constructs.

1. A finite set P of *predicators*, or roles.
2. A nonempty set O of *object types*, containing *label types* and *entity types*. Label types are concrete object types (such as Name), while entity types are abstract (such as Person). Typically, entity types are connected to label types by binary relationships called *bridge types*. Officially, there may be no entity type without such relation; however, we will omit many of them in our model for the sake of clarity.
3. A partition F of the set P; elements of F are called *fact types*. Bridge types are a special kind of fact type. Also, fact types may be objectified and play the role of an object type.
4. A set G of *power types*, a special kind of object type. A power type is the parent of its element object type. Instances of a power type are always (nonempty) *sets* of instances of its element type. A power type does not necessarily contain all possible sets.
5. A set S of *sequence types*, which are quite like power types. The differences are that sequence type instances may contain an instance of its element type multiple times, and ordering is important. As such, we call an instance of a sequence type *tuple*, rather than set.

6. A binary relation Spec on object types, capturing specialization. *A* Spec *B* means: the instances of *A* are formed by all instances of *B* meeting some requirement. This requirement is referred to as the *subtype defining rule*.
7. A binary relation Gen on object types, capturing generalization. *A* Gen *B* means: all instances of *B* are also an instance of *A*. Typically, different object types are generalized towards one common parent object type. As such, all instances of *A* must be identifiable by their properties as an instance of *B*.
8. The function Fact : $P \rightarrow F$ yields the fact type containing given predicator.
9. The function Base : $P \rightarrow O$ yields the object type associated to given predicator.

Using the elements above, a formal grammar may be constructed. In this grammar, object types are grammatical concepts and fact types are language rules. It enables expressing elementary and derived facts about the shadowed world, in our case the welfare environment. Furthermore, we express restrictions upon the allowed population of the model by using the usual ORM constraints.
Like ORM, PSM uses a graphical notation of which the above set model is the formal basis. We will use this graphical notation in the design of our model. We assume reader familiarity with ORM graphical notation. A complete overview of PSM as well as elaborated examples of both the mathematical linguistic and graphical notations can be found in [7].

2.3.2 Object Role Calculus

Besides creating a fundamental network of elementary facts in our ORM model, we would also like to express things about the welfare domain that do not have an elementary nature. Frequently, these expressions are combinations of multiple facts, combined with logic operators or conditions. To express such facts in a formal and exact way, we would traditionally have to use abstract languages from mathematics. However, such languages are typically hard to understand for people inside the target domain. The main argument for using ORM still holds: they are used to natural language and a more specific vocabulary about welfare related concepts. This vocabulary is found in the ORM schema, and therefore it is logical to extend the ORM vocabulary with some logical keywords that have sufficient expressive power to formulate interesting facts in structured natural language, but are also sufficiently formally defined to facilitate disambiguous reasoning. Such extension has been proposed by Ter Hofstede et al. together with PSM, and was originally called LISA-D [7]. Later, the language has been slightly refined and renamed to Object Role Calculus (ORC). There is a more obvious link with Object Role Modeling in this name. We will shortly summarize ORC here; for a complete overview we refer to [5] and [7].

We will mainly use the following constructs from ORC. In the following definitions, we use D_i for an arbitrary ORC expression, analogously to [5].

1. LET D_1 BE D_2. This defines a new concept (D_1), based on a combination of already present concepts (D_2). D_2 may be constructed directly from the PSM model or

from earlier ORC definitions. For example, we might introduce an alias for a known PSM object type *Product*: LET Deliverable BE Product.

2. *D_1 D_2* THAT *D_3*. With this construct, we can express a correlation rule. This is a combination of multiple properties of the same object type. For example, we might write Product delivered at Location belonging to Organization producing THAT Product. Omitting the keyword THAT would result in the selection of Products that are delivered at a location belonging to an Organization producing *any* Product.
3. *D_1 D_2* ANOTHER *D_3*. Analogous to THAT, but now we state explicitly that the *D_3* instance must be different from *D_1*, rather than equal.
4. *D_1* AND *D_2*. Operator to express that *D_1* and *D_2* should both hold.
5. *D_1* OR *D_2*. Operator to express that either *D_1* or *D_2* should hold.
6. *D_1* BUT NOT *D_2*. Operator to express that *D_1* should hold, but *D_2* should not.
7. *D_1 D_2* IS ALSO *D_3*. With this construct, we state that when relationship *D_2* holds for instances from *D_1*, relationship *D_3* should also hold. For example: Product delivered in Area “6500” IS ALSO delivered in Area “6600”.
8. *D_1* IMPLIES *D_2*. This is the subset condition again, but now it concerns both ends of expressions *D_i*. Whenever *D_1* holds, *D_2* should also hold. For example: Product delivered in Area IMPLIES Product delivered at Address at Location in Area.

Since ORC is used to formulate derived fact types, we will only scarcely use these constructs in our case study to illustrate its purpose. Using the full set of ORC keywords, any desired rule about the basic domain model may be expressed based on the grammar provided by the ORM model. This enhances the number of natural language concepts and relations that are covered by the domain model, without introducing redundant ORM constructs.

3 Welfare Environment Model

It's important to note that there is no such thing as *the* welfare environment. We are trying to capture some part of the real world, which consists of entities like persons, organizations, products, relations. The definition of which part this is, is not one we will find in a dictionary, nor in literature. It is a subjective and rather arbitrary definition, usually formalized for a specific purpose, if formalized at all. However, we would still like to reason about it. Therefore, we performed a case study to create a model of the welfare environment as experienced by MEE.

We define this welfare environment as follows: *"all organizations, products, services and information related to needs of individuals who are impaired in some way"*. This is what we will try to capture in our model. For more details, we refer back to section 1.1.2. To maximize general applicability and for the sake of clarity, we focus on the most important concepts and relations.

This chapter is organized as follows: in section 3.1, we gradually introduce all relevant concepts, resulting in an ORM model and ORC rules. We follow up in section 3.2 with an analysis of this model, providing an overview of interesting possibilities and challenges.

3.1 The model

In this section, we will look at all relevant concepts inside the welfare environment, as well as those directly connected to it. Looking at Figure 2, we can identify two key parts of our model.

1) The relevant welfare-related information known by a MEE counselor, which reflects the actual environment. This is the knowledge captured by the Social Map. As the environment is structured, so is this information. This part of the model defines the *solution space* available to the client directly or through a counselor.
2) The connection to the client; where we look at what a typical question looks like and how it can be answered using facts in the welfare environment. In this part of the model, we investigate the characteristics of the *problem space*. We will add basic reasoning and model the dynamics of interaction.

The combined model is a reflection of the total information flow in typical welfare problem solving. But first, we will start off simply with the abstract core of the model.

3.1.1 Model core

One of the most universal and abstract facts is also a fact within the welfare environment. The shortest abstract of many academic writings is the core of our model:

[ORM1] Solution *X* solving / solved by Problem *Y*.

This is our first structured natural language sentence from the welfare domain. Note that we use the shorthand notation of sentence interleaving, writing down a bidirectional sentence in one go. It should be read as the sentence pair "Solution X solving Problem Y" and "Problem Y solved by Solution X". Together, they can be transformed directly to an ORM fact type, with two objects ("Solution" and "Problem") and two roles ("solving" and "solved by"). See Figure 7 for this simple ORM snippet.

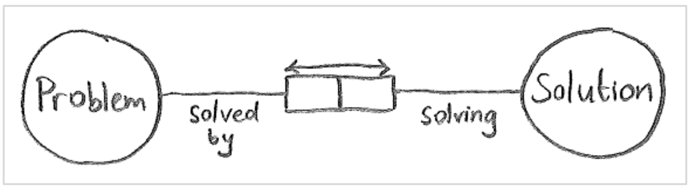

Figure 7 – Problem and Solution

The uniqueness constraint (UC), indicated by the double-sided arrow, is on both roles. This indicates that Problems may have multiple Solutions, and Solutions may solve multiple Problems. We can regard this core model as the interface between the upcoming solution space and problem space. We will elaborate both abstract concepts now, starting with the Solution concept. The Problem concept will be covered in section 3.1.3.

3.1.2 Solution space

We will introduce relevant concepts around the abstract Solution concept step by step, creating different ORM snippets. Together, they form the solution space. We start off by refining the Solution concept itself.

3.1.2.1 Redefining the Solution concept

The first question we ask is: "What is a Solution?". To answer this question, we take a look at the welfare environment. Inside the welfare environment, there are companies, governmental and non-governmental organizations and individuals who have something to offer for people with a specific welfare need. In structured natural language, this may be represented as:

[ORM2] Solution : Organization *X* offering / offered by Product *Y*

This introduces a new fact type, connecting the new concepts Organization and Product. This fact type is very suitable to be objectified as a refinement of the Solution concept. Concrete problem solving usually happens when a client gets in contact with an appropriate Organization, which has something to offer for him (i.e. the Product). The most accurate definition of a Solution is the combination of Organization and Product. In example sentence ORM2, we define the Solution concept to be the objectification of the fact type following the semicolon. We will use this notation multiple times in this chapter when introducing new objectified fact types.

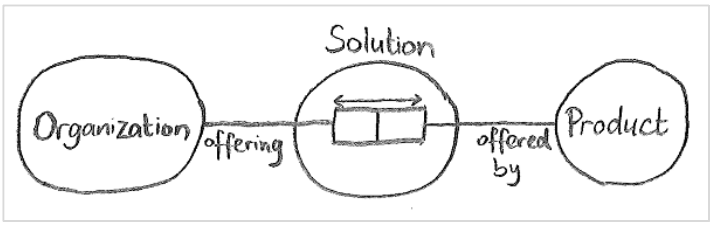

Figure 8 – Refining the Solution concept

In Figure 8, the new fact type and redefined Solution concept are shown. Note that throughout the creation process, we are focusing on new model parts in the shown snippets, omitting unchanged parts for readability. Although the omitted part of the model is not shown, new parts are always an extension or refinement upon what we already had.

The UC is again on both roles. This means that Organizations and Products may both also play a role in other Solutions. For example, the Problem "I cannot walk" may be solved by some wheelchair offering. Suppose that the "Roundabout 2000" is such a wheelchair; it would be an instance of the Product concept. Wheelchairs of this type may be offered by different Organizations. Also, a wheelchair delivery organization can surely be expected to deliver different types of wheelchairs.

It's important to note that the Product concept as intended here has a very abstract nature. Therefore, we will leave it as an atomic entity type, although Products surely have relations other than their role in a Solution. For example, we might think of a manufacturer, build year or dimensions. Although those properties are suitable for a wheelchair, they don't make sense when talking about another valid Product: "basket ball training". In case of that product, we would rather add fact types for a trainer and course times. In our model, we try to keep the concepts as concise and general as possible to be able to focus on the key points of interest in the welfare domain. A real life application (in the Semantic Web) would deal with more details. These details may be introduced by further extending the Product concept just like we are extending the Solution concept now. One of the most straightforward ways would be to identify and model the characteristics of different product categories and generalizing these categories to the more abstract Product entity type as it is now.

3.1.2.2 Organization

When we look at Organizations, there are a few generally applicable related concepts worth investigating. More often than not, the site of an organization is an important detail in the search for an applicable solution. Therefore, we take this detail into account. Exemplary structured sentences about the housing of an Organization are as follows.

[ORM3]	Organization *X* housed at / housing Site *Y*
[ORM4]	Organization *X* having headquarters at / headquarters of Site *Y*
[ORM5]	Site *X* having role / role of Siterole *Y*

The first two sentences (ORM3 and ORM4) are about organization housing. Typically, an organization has at least one site where it is reachable for one or more kinds of activity. Especially larger organizations with multiple sites have one site designated as headquarters. However, both fact types regarding a Site do not explicitly require involvement of every Organization. Theoretically, an Organization may exist which has no physical site related to it. Sentence ORM5 expresses the role a Site plays. Typically, different activities are related to different Sites, especially in larger Organizations. Examples of Site roles include postal, delivery, visitor or production. The Siterole concept contains all occurring activities that may be related to a Site. Putting all this into an ORM diagram, we get the following:

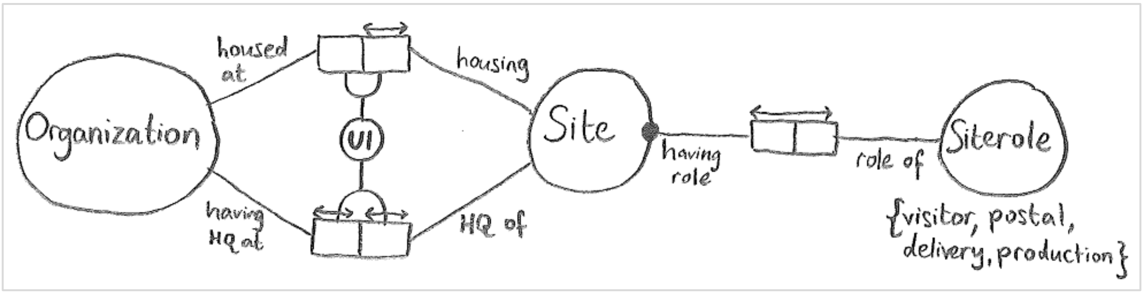

Figure 9 – Introducing the Site concept

Every Site should have at least one Site Role assigned to it. This is enforced by the totality constraint (TC), depicted by the large dot on the *"having_role"* role. The population of the Site Role concept itself may only be taken from a set of predefined values. In Figure 9, there is a *value constraint* containing four possible values on the Site Role concept. These values may seem awkwardly limiting, but they facilitate the expression of automatic reasoning rules about different Sites of an organization. For example, suppose we would like to know where to send a letter to Organization "MEE". We need only use the following ORC sentence:

[ORC1] Site housing *"MEE"* AND ALSO having role *"postal"*.

Any Site satisfying this rule is a valid MEE site to send a letter.

Notice the lack of a totality constraint on both Organization and Site, regarding the office and headquarters. This is because not all Organizations need to have a (physical) site, and not all relevant sites need to host an organization. For example, a site may just be a general place of delivery. The UC for the office fact type is only on the Site role: a Site is forbidden to house two different Organizations. This distinguishes the abstract Site type from its physical location. Multiple Sites may have the same physical location, but by this UC we enforce that we are able to address them all separately when it comes to their Site Role.

Headquarters are defined using ORM rather than ORC. Specifically, we use the subset constraint to enforce that any instance of headquarters is also an instance of office. Additionally, an organization may only have one headquarters. This is enforced by the additional single role uniqueness constraint.

3.1.2.3 Delivery methods

Now that we have a notion of Sites related to Organizations, we introduce *delivery methods* of a Solution. We should keep in mind that "delivery" should be interpreted in the broad sense, for it is implicitly related to the Product part of the Solution. Recall that both the wheelchair and the basketball training are Product examples which need to be "delivered". Although "delivering training at the gym" has an illogical sound to it, it is logically sound. Should the use of natural language ever drive us nuts, we can use ORC to define a convenient alias.

Solutions may be delivered in two different ways: at a fixed location or at home. Here are the corresponding structured natural sentences:

[ORM6]	Solution *X* delivered at / having delivered Site *Y*
[ORM7]	Solution *X* delivered in / eligible for delivery of Area *Y*

Although both methods of delivery are equal in some sense (the Solution arrives at some other place), they are also different. Take a look at Figure 10 below.

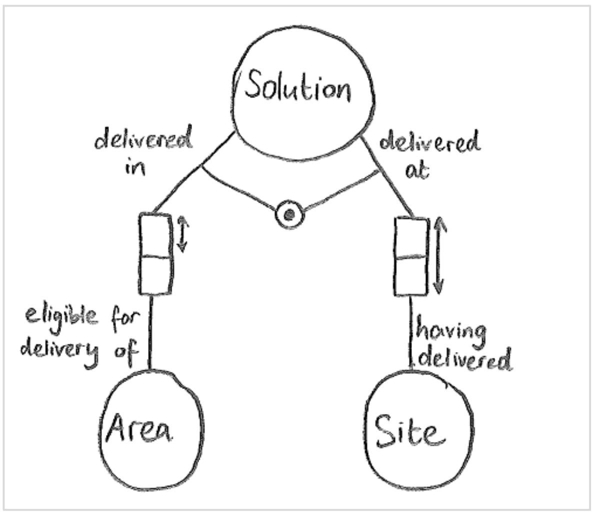

Figure 10 – Solution delivery methods

Delivery at location (ORM6) relates a Solution to a Site, the concept we saw before. A Site is a single fixed location. Furthermore, we have some information on what role the site fulfills. When designing a system based on this model, we could for instance formulate a reasoning rule about a solution to be delivered to all sites related to its organization, but only if that site has the "delivery" role. This kind of rule may be specified according to the needs of specific organizations and doesn't need to hold system-wide. Therefore, there is no such harsh restriction in this ORM model: any site may principally host a delivery at location. With ORC, we might add the rule like this:

[ORC2] Site having_delivered Solution IMPLIES Site having_role *"delivery"* AND ALSO housing Organization INVOLVED_IN Solution

Sentence ORM7 concerns delivery of a Solution in a geographic Area. This is a new concept, which we will soon further refine. For the time being, it's an abstract type which stands for the area to which the Solution is available. Typically, this means that it is something to be used at home. As we can see, there is an UC on only one role here; contrarily to the on-site delivery a Solution is just deliverable to one area. Of course, what this area looks like may geographically be very hard to describe. Using ORC, its characteristics are captured very simply; we can define a Service Area as follows.

[ORC3] LET ServiceArea BE Area eligible_for_delivery_of Solution

A ServiceArea is a specialization of a regular Area. ORC3 is its subtype defining rule.
As a last note, we see a TC over two roles here, enforcing that any Solution is at least deliverable in one of these two ways. After all, what use is a solution if it cannot be "delivered" to solve a problem?

3.1.2.4 Geography

Geography is an important factor in the determination of an appropriate Solution for a client's problem. The purpose of our model is to get a physical client in touch with a physical solution. Adding geographical concepts to our model places the abstract concepts we saw up until now into the real world. It's of great general value to be able to reason with geographic concepts in the model, just like we are used to in real life. For example, knowing that a country is larger than a street and being able to express an area of delivery in terms of postal codes, but also in terms of geographic coordinates. Maybe we would even like to express a vague definition like "near Nijmegen" as a delivery area. When we have a conceptual framework grasping basic geographic concepts, we can use ORC to express this kind of derived concept based on specific needs. This setup is analogous to the one we saw in the previous section, where we chose to keep the ORM model generic, using ORC rules to narrow down constraints when desired (see ORC2).

The geography model consists of two parts. The first part is modeling *locations*. built up around an abstract concept called "Location". This is simply any physical location. In practice, it will mostly be a combination of two geographic coordinates. They might for example be coordinates in the well-known latitude-longitude system. However, we do not make such implementation choices in our domain model and leave the Location concept atomic. The Area concept we saw earlier is just a set of these Locations.

[ORM8] Area CONTAINING Location

To distinguish power type definition from fact type definition, we use the keyword CONTAINING. ORM8 defines Area to be a power type having Location as its element type.

The second part is modeling *geographic concepts*. By this, we mean concepts used in natural language to indicate a specific geographic area. Examples of this are "address", "postal code", "street", etc. We regard an Address as the center concept of this part. It consists of a postal code and a (house) number.

[ORM9] Address : Housenumber in / with PostalCode

[ORM10] Site *X* at / of Address *Y*

An Address is located at exactly one location, and every Address has a Location. This ties both geographic model parts together.

[ORM11] Address *X* at / of Location *Y*

The following ORM diagram shows all new concepts and relations graphically.

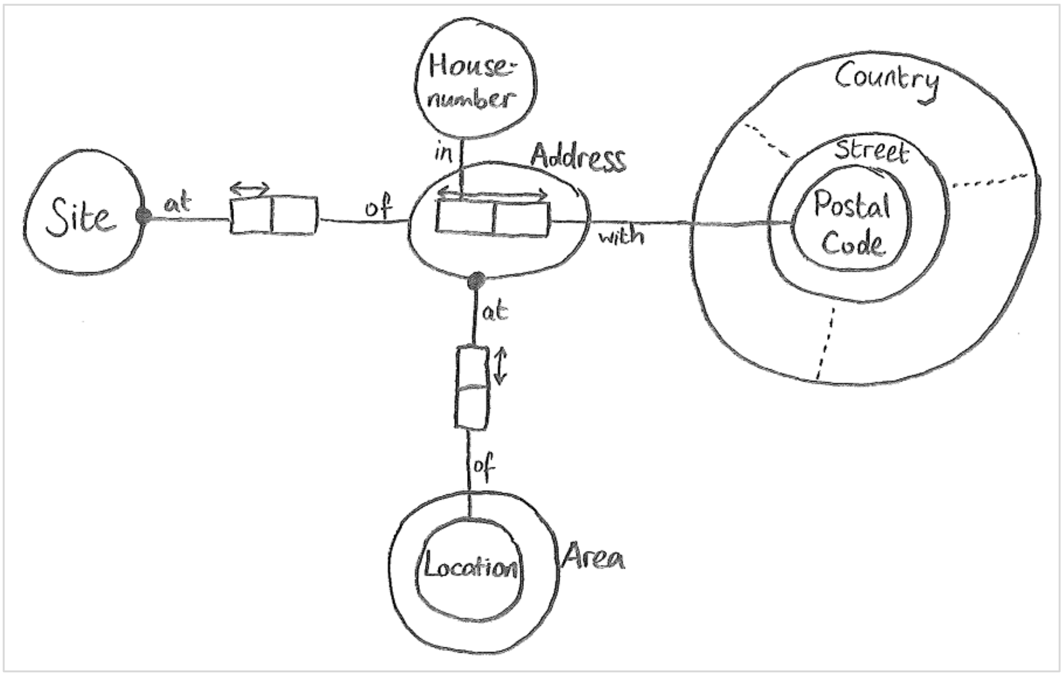

Figure 11 – Geography model

Note that we placed the PostalCode concept inside a layered model of different geographic concepts. Each geographic area concept is a power type of a smaller geographic area concept. We defined a PostalCode to be the elementary area concept, followed by Street, finally leading to the largest concept: Country. This model part is a basic example of how geographic concepts may be integrated into the welfare environment to be able to reason with or about them. In a real application, we could attach a more complete semantic web of geography, linking more geographic terms to each other, with a more elaborated relationship. However, it's already possible to structurally store all postal codes and larger geographic areas in this model.

This geographic model allows us to answer relatively exotic queries. For instance, we can answer "Is Solution X delivered at Toernooiveld 1, Nijmegen?" using a fully populated model and some basic reasoning. The reasoning system should understand that Nijmegen is a City containing the Street Toernooiveld, which is a set of Postal Codes. Only one of them has the number 1 attached to it, leading to a unique Address instance. Now we only need to verify that the Location of this Address is inside the Service Area of Solution X. Strictly, we are cheating a bit here. In our model, nothing enforces that only one PostalCode within a Street is associated with a HouseNumber, as is a geographic rule. This may be added using ORC and is formulated as follows:

[ORC4]	Street CONTAINING PostalCode with HouseNumber BUT NOT Street CONTAINING ANOTHER PostalCode with THAT HouseNumber

3.1.2.5 Refining Solution delivery

Solutions are not always just delivered under all circumstances. More often than not, there are conditions to their availability for a client. For example, a Solution might only be deliverable to *women* experiencing the Problem it solves. Although a certain male might have the exact Problem that is solved by this Solution, he will still not be able to receive it because he doesn't meet the condition "Client needs to be a woman". This leads us to refine our general model, adding the concept DeliverableSolution.

[ORM12]	DeliverableSolution : Solution X deliverable when being met / enabling delivery of Conditions Y
[ORM13]	Conditions CONTAINING Condition

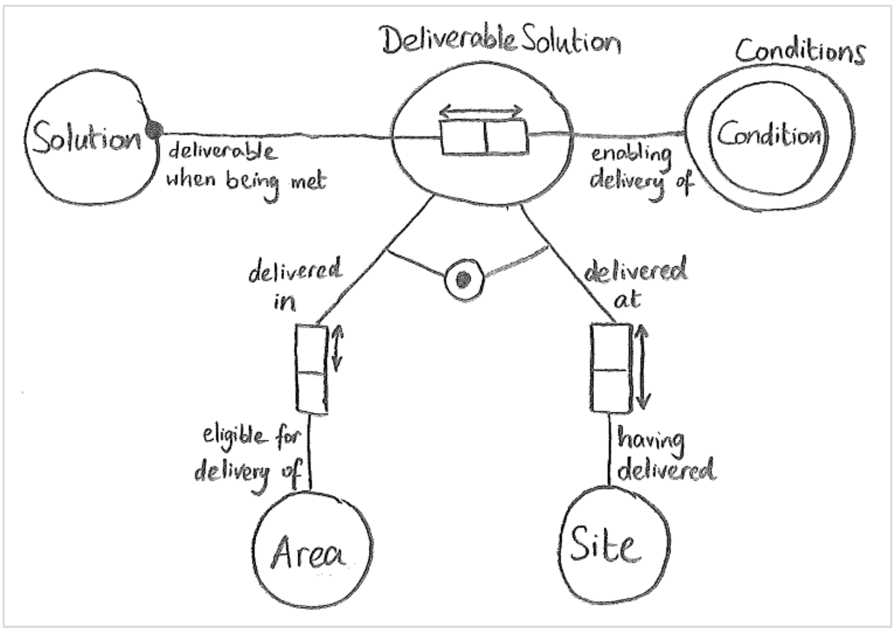

Figure 12 – Adding the DeliverableSolution concept

This change also impacts the model part we constructed in section 3.1.2.3. The delivery methods did not change, but they should now be connected to the DeliverableSolution concept. In Figure 12, the new concepts are added, including relevant constraints.

Note that the Condition concept is very abstract. The nature of conditions may vary heavily, therefore it is difficult to further refine the Condition concept. Just like we argued when dealing with the Product concept, we might also cover various forms of condition by elaborating some and then generalizing them to the abstract Condition concept. Because conditions are a more vital part of client – counselor interaction, we will investigate one such concrete form: the *age restriction*.

<u>The age restriction</u>
The age restriction is an example of a condition that is related to some Solutions. When modeled, it may be automatically verified whether a client satisfies it. The following example sentences are expressing relevant facts. Figure 13 shows them graphically.

[ORM14]	Person *X* having / of Age *Y*
[ORM15]	AgeGroup CONTAINING Age
[ORM16]	AgeGroup *X* targeted by / targeted at Solution *Y*

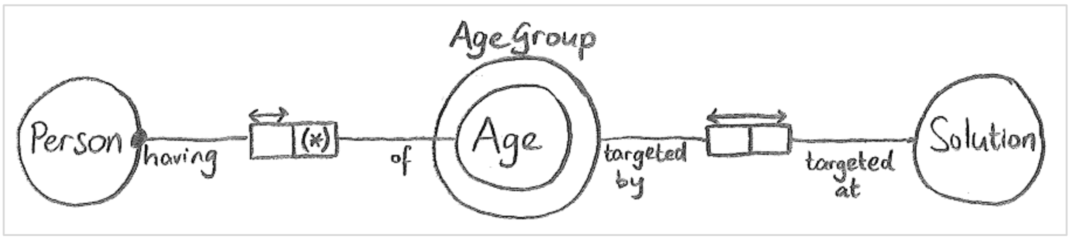

Figure 13 – Example of a delivery condition: age restriction

Note the asterisk sign (*) on the Age-of-Person role in the ORM model. This denotes that Age is a *temporary variable,* which will be computed when needed based on static data. Assuming we can compute the Age of a Person, it can be automatically verified whether a Person is eligible to receive the Solution targeted at some Age Group: the Age of the Person only needs to be a member of that set. This condition may be expressed in ORC.

[ORC5]	Client having Age IN AgeGroup targeted by Solution

When this condition is associated with a Solution, it results in a DeliverableSolution which is deliverable to its associated delivery Area and/or Site when this condition is satisfied.

Another example of an automatically verifiable delivery condition is the *gender restriction* we saw in the example with which we started this section off. Creating a model for this condition is more straightforward than the age restriction. We leave it as an exercise for the reader.

3.1.2.6 Textual knowledge

The presented collection of models defines an overview of the basic composition of the welfare environment, when put together. It defines the core of what basic concepts and relations are available to answer the needs of a regular client, who is searching for a solution to a certain problem. Within this model, we can identify a special kind of client. He is just searching for information about something welfare related. This information might serve different purposes: for example it provides insight in his situation or what possibilities there are for him. It might lead to spawning new interests or guide the client in an unstructured way. Therefore, it does not necessarily fit the basic definition of a client being a Person with a Problem. However, textual knowledge is a vital part of the welfare environment, and it is treated as such in the current Social Map approach.
The core concept of this informative part of the environment is simply an unstructured text called a Describing Text (or shortly: DText). The concrete information is inside these DTexts; due to the heavily unstructured nature of its content, we cannot further conceptually break up a DText itself. However, there are interesting things to say about the structure at a higher level. We regard a DText as an elementary concept which is targeted at providing information to a human. DTexts are structurally related to one another through two external structuring mechanisms: a *thesaurus* and a *DText Tree.* Both mechanisms are linking DTexts, so they have a navigational and relational purpose. We will look at both mechanisms in more detail.

Structuring DTexts by means of thesaurus terms
A thesaurus is a semantically ordered vocabulary and as such a structured set of terms. These terms may be designated to a DText, typically functioning as a very short summary or providing hints about the content of the text. This relation may be used to find other DTexts (or other objects for that matter) related to this thesaurus term, semantically connecting different pieces of information or concepts from the welfare environment. Specific welfare environment thesauri have already been designed. In our case, the Dutch NIZW thesaurus is being used[8].

Structuring DTexts by means of a DText Tree
Besides linking DTexts to thesaurus terms and making use of its semantic structure to relate different items to each other, we might also structure them directly. In fact, this is a quite common feature, comparable to a text book that can be navigated by its index and its table of contents. Both ways of navigation may be preferable at different times. Generally, someone who is in need of information first needs a rough guide to clarify what he is really looking for, while the result of this action may be fine-tuned afterwards. Note that both the table of contents and the index of a text book are strictly redundant once someone is able to directly search by any keyword, as is common practice on the internet. Describing Texts are manually structured in a navigable tree with labeled nodes which act as categories. By selecting applicable categories, one may

[8] http://www.thesauruszorgenwelzijn.nl/

refine his interest area until the best matching DText is found. We will now analyze the composition of a DText Tree.

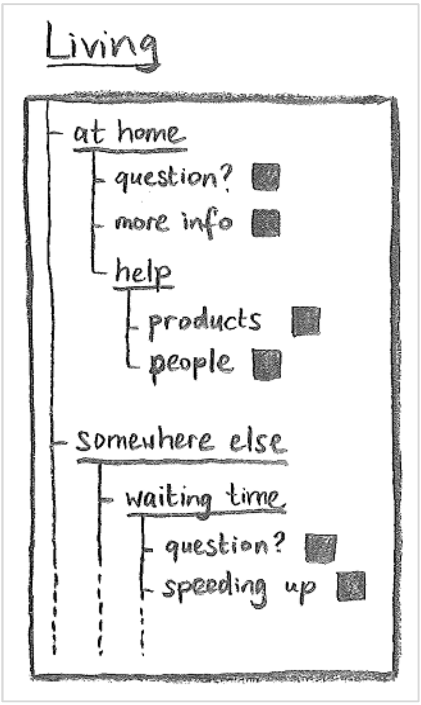

Figure 14 – Example DText tree

Figure 14 shows an example of a DText tree about Living. It contains various texts with information about this general subject. We call "Living" a Main Category. After selecting this main topic of interest, we can see how it is divided in different subcategories. In this case, we see "at home" and "somewhere else" as examples of first level subcategory and that there is also a second level inside both. Of course, there is no boundary as to how many levels there may be, so we just call them "Category" regardless of level. In the example figure, Categories are underlined. Last but not least, we have the actual Describing Texts are depicted with labeled dark squares; a combination of Title and DText. Essentially, Titles do not differ from Categories, for they are both fulfilling the same navigational and presentational role. The difference is that a Category labels a tree node, while a DText title labels a tree leaf. Since both nodes and leafs are some part of a tree, we call them both a DText Subtree. Altogether, this leads to the following formal rules.

DText Tree composition

[ORM17]	DTextTree SEQUENCING DTextSubtree
[ORM18]	DTextSubtree : DTextNode OR DTextLeaf
[ORM19]	DTextLeaf : Title *X* labeling / labeled by DText *Y*
[ORM20]	DTextNode : Title *X* labeling / labeled by DTextTree *Y*

Thesaurus link

[ORM21]	DText *X* described by / describing ThesTerm *Y*

Additional concept definitions

[ORC6]	LET Root BE (DTextNode BUT NOT OCCURRING-IN DTextTree)
[ORC7]	LET Subject BE (Title of DText)
[ORC8]	LET Category BE (Title of DTextTree)
[ORC9]	LET MainCategory BE (Category INVOLVED-IN Root)

We defined four additional concepts as a subtype of the general concepts. They are common terms used in the textual knowledge representation and have an illustrative purpose. A Root node is a node that is not occurring as an element of a larger tree (ORC6). With the concepts Subject and Category, we distinguish between title labels associated with an actual text (ORC7) and title labels associated with a tree, thus used for navigational purposes (ORC8). Finally, a Category of a Root node is a special kind of category. With rule ORC9, we may address it directly as a MainCategory. All new ORM and ORC rules together lead to the model depicted in Figure 15. Note that this model is independent of the model we had constructed until now, because it models a different aspect of the welfare environment.

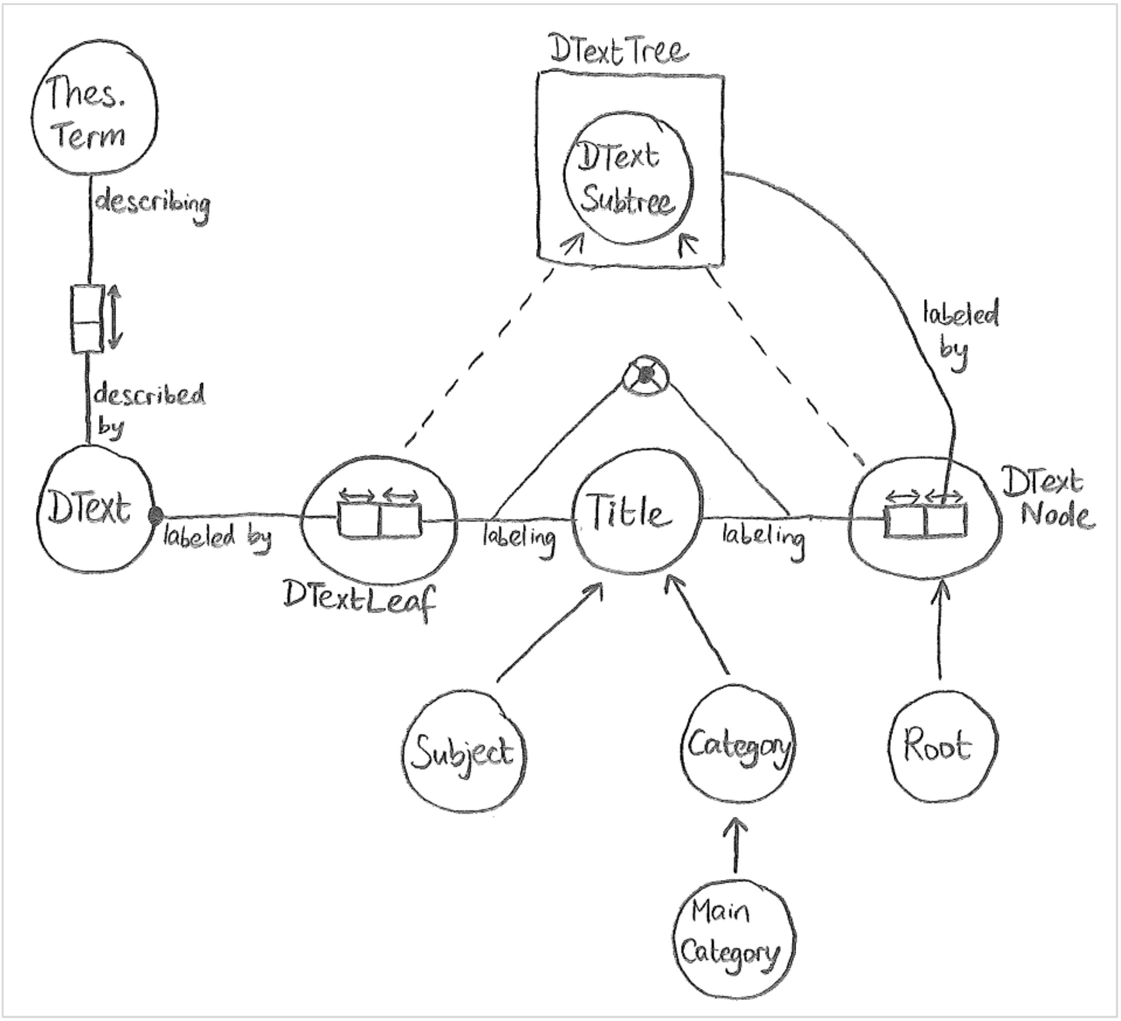

Figure 15 – Describing Texts about the Welfare Environment

3.1.3 Problem space

Now that we have explored the Solution concept, we will proceed with the Problem concept. The importance of this concept immediately becomes clear when we formally define the Client concept.

[ORC10]	LET Client BE Person experiencing Problem

The Problem concept has a crucial role in the subtype defining rule distinguishing a Client from any other Person. Note that since we are modeling the welfare environment, a Problem is always regarded to be welfare-related.

Inside the welfare domain, there are Solutions that solve Problems, as we saw in the previous section. However, it is very unlikely that there is a direct match between the Client's Problem and the Problems that have a Solution. In a naïve approach, this would mean that the counselor would have to send the client back home without a cure to his pain. Luckily, we are not naïve, and we will look more closely to what a Problem essentially is.

We define a Solvable Problem as a Problem that has a Solution:

[ORM22]	SolvableProblem *X* being solved by / solving Solution *Y*

Sentence ORM22 replaces sentence ORM1. Now, how does the Problem concept relate to the SolvableProblem concept? A client's Problem is solvable when it is a SolvableProblem or it can be regarded as a series of SolvableProblems, where solving each of them individually also solves the Problem as a whole. Identifying SolvableProblems is the task of the counselor, and possibly also of the ultimate intelligent supporting application. The result of this process of analyzing a Problem to identify Solvable Problems inside of it can be represented using a pie chart. The examples in Figure 16 illustrate possible results of a counselor's analysis of a Problem.

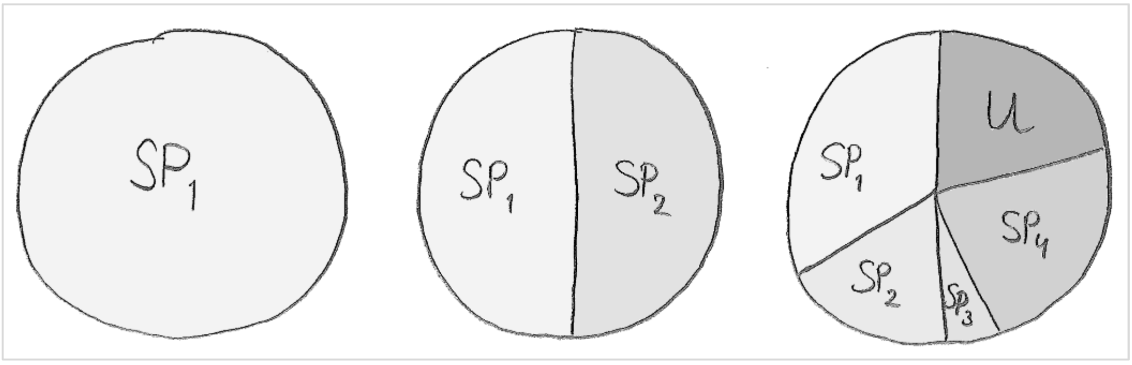

Figure 16 – Different breakups of a Problem to SolvableProblems

SP_x denote different SolvableProblems which constitute part of the total Problem. Also, part of the Problem may be Unsolvable (U). Note that Unsolvable just means that the counselor doesn't know any Solution for it. We follow the *closed world assumption* here, which means that if we don't know a Solution, we boldly state that "... therefore there isn't any". How a Problem is decomposed into a pie chart like the ones above is a very interesting process and it would be worthwhile to have the information system help the counselor as much as possible with this. In this model, the main question a counselor – and therefore the information system – is trying to answer is: *"**What is the largest possible part of the Problem that can be solved using only Solutions of which the Client can meet their delivery Conditions?**"*. The ORM model capturing this is our target in this section.

Let's take a look at an example of a typical counseling meeting. Alice, a young woman of 25 years old, just revalidated from a car crash accident. Unfortunately, she will have to live in a wheelchair from now on. She used to be the sporty type, but what should she do now? One day, she decides to visit a welfare counselor specialized in this kind of questions. The counselor has a meet and greet with Alice and he gains an understanding of her situation in a short session. There are all kinds of organizations, products and services targeted at practicing sports with a disability, but a lot of them have special requirements. For example, there is a gymnastics group targeted at elderly people, a target group Alice clearly cannot identify herself with. Some training courses require prior knowledge, or a health club gives preferential treatment for people who live in roughly the same area as its location. The seemingly simple task of guiding Alice towards a new life with sports is not that simple after all. After a few sessions, Alice gains a fine understanding of her possibilities and is able to choose from amongst them.
So how does this example break up? We will now reformulate it in terms of our conceptual structure and see what's added.

1. Person *Alice* experiencing SolvableProblem *P*
2. SolvableProblem *P* solved by Solution *S*
3. Solution *S* deliverable when being met Conditions *C*
4. Condition C_i IN *C* IS ALSO met by Person *Alice*

The challenge to be solved is to find the best possible set of SolvableProblems adhering to these four rules. The first step is to identify all SolvableProblems being a part of Alice's Problem. However, this is more difficult than it seems. Not all SolvableProblems known to the counselor are also solvable *in Alice's case*. Constructing a set of SolvableProblems for Alice is therefore twofold. First, we need to validate whether Alice meets Conditions C for all relevant general SolvableProblems. After that, we should choose a subset from this set which will altogether solve the largest part of Alice's Problem. This way of working is not necessarily the way a counselor exactly acts. However, it captures the essence of the difficult matching problem that needs to be solved. The ORM model in Figure 17 shows the counseling process. We added the following sentences:

[ORM23]	Person *X* experiencing / experienced by SolvableProblem *Y*
[ORM24]	Person *X* meeting / met by Condition *Y*

We included the original sentence "Person experiencing / experienced by Problem", since that essentially didn't change. The dashed line between Problem and

SolvableProblem indicates the interpretation process involved, which we cannot model with a simple power or sequence type.

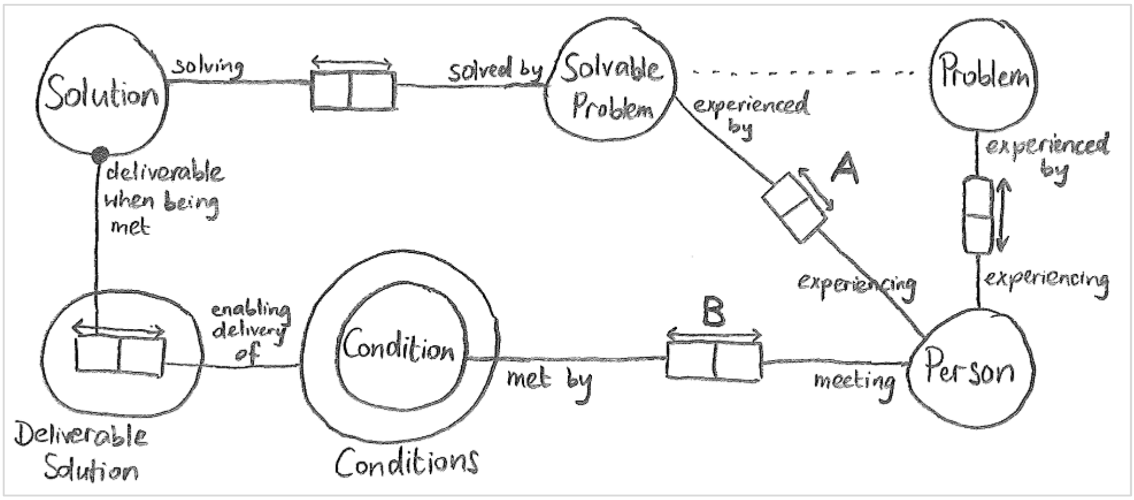

Figure 17 – The counseling process

The addition of the Client to the static solution space introduces two new fact types labeled A and B. Figure 17 may be regarded as a dynamic information system which needs population. The counselor starts this population process by analyzing the Client's Problem, thus filling fact type A. This acts as a filter upon the static knowledge base; all Solutions can be retrieved matching with the identified Solvable Problems, and the related Condition sets may also be populated through related Deliverable Solution instances. The result of this population is a list of identified Solvable Problems and their Conditions for delivery. Now for the hard part: all that remains is calculating the set of Solvable Problems with an *actually deliverable* DeliverableSolution that solves the largest part of the original Problem. For this, fact type B needs to be populated. As we saw at the introduction of the Condition concept, automatically verifying whether a condition is satisfied is not straightforward. Correctly populating fact type B ensures that the Solvable Problem concept is finally only populated with problems that are actually solvable *in Alice's case*. Although the population of this schema is far from trivial, the population does provide a complete overview of the Client's situation and what possibilities there are for her.

3.2 *Possibilities and challenges*

So far, we have focused on the more traditional part of (ORM) information system modeling. We paid attention to those concepts and relations that are easily translated to a data model. Storing data is most easy when it is static (i.e. doesn't change at all or not often). We just need to fill the generic model with instances from the real welfare domain, and the information system may thereafter be queried based on users' need. However, not everything in the welfare environment is static. To illustrate the counseling process using our model, we will first revisit the running example in section

3.2.1. After that, in section 0 we will zoom in on different interesting possibilities and challenges arising in our model, which come forth from dynamic aspects in the model.

3.2.1 Alice revisited

An information system based upon our domain model contains a static population which describes all possible (Deliverable) Solutions, which are closely related to a Solvable Problem and a certain set of Conditions, as was shown in Figure 17. A sample population may be as follows:

Solvable Problem	Deliverable Solution		
	Solution		Conditions
	Organization	Product	
Sports	Gym A	Youth Basketball	Age between 18 and 30
Sports	Gym A	Basketball 55+	Age >= 55
Sports w/ wheelchair	Gym A	Disabled Mothers' workout	Age between 20 and 45; Must have at least 1 child; Must be a woman
Sports w/ wheelchair	Gym B	Wheelchair Hockey	Living max 20 km from Gym B
Wheelchair usage	Gym C	Training course	
…	…	…	…

Table 1 – Solution space population

In this sample information system fragment, we suppose the counselor already identified that Alice's Problem has something to do with getting to practice sports in her wheelchair. However, to solve (part of) this Problem, different Solvable Problems may be available. As we see, there is a Solution for "Sports" in general, but also for "Sports with a wheelchair" in particular. Also, Alice may be assisted in general "Wheelchair usage", not specifically targeted at sports but without any conditions. Although not shown here, the counselor typically knows more details about the surroundings of a particular Solution. For instance, he may know that Alice's Problem is solved most effectively by either solving "Sports w/ wheelchair" or solving both "Wheelchair usage" and "Sports", because in a regular – though disabled-friendly – environment, affinity with one's wheelchair is assumed and there are less other disabled people to help getting used to it in the sports environment. However, not all listed Solutions are really deliverable to Alice, because she doesn't fit all conditions. Alice's age is 25, she doesn't have any children and her home is 19 km from Gym B. This results in two options for her, outlined in Table 2. Alice will have to choose between Wheelchair Hockey in Gym B, which will accept her but is 19 km away, or playing regular basket ball assisted by a wheelchair usage training course. This will involve two gyms, but they are closer to her home. And of course, hockey is not the same as basket ball. It's up to Alice to choose between these two options.

Solvable Problem	Deliverable Solution		
	Solution		Conditions
	Organization	Product	
Sports w/ wheelchair	Gym B	Wheelchair Hockey	Living max 20 km from Gym B
Sports	Gym A	Youth Basketball	Age between 18 and 30
Wheelchair usage	Gym C	Training course	

Table 2 – Possible breakups of Alice's Problem with associated Solutions

As we can see, retrieving the relevant part out of the static information system after attaching a dynamic Client concept proves difficult and requires a lot of human thought, particularly by the counselor but also by the Client herself. The non-trivial parts are found at the analysis of what Solvable Problems lie inside the Client's Problem (fact A in Figure 17) and determining whether the Client fulfils associated Conditions (fact B in Figure 17). Even after this, it may still not be clear what combination of Deliverable Solutions is most desired by the Client. In our simple example, choosing between hockey and basket ball may already do the trick, but most often the choice will be harder (and thus automated machine assistance in choosing will be more difficult). We will now briefly analyze some important challenges concerning dynamic data. Addressing these challenges in our model is beyond our scope, although they should be dealt with in a real application.

3.2.2 Dynamic knowledge

We will now look at some examples of what we call *dynamic knowledge*. This is knowledge that is not fully static because it depends on a realtime component. We will look at some of these components: time, interpretation and environment.

3.2.2.1 Time dependency: Age

A Client's Age is a concept easily grasped by humans. However, it is constantly changing. The Age concept can be split up in a static part (birth date) and a dynamic part (current date). Just subtracting someone's birth date from the current date is enough. A process that's quite easy, really. But in a static model, it is difficult to just add an Age concept directly because it would not be possible to populate it statically. To avoid this problem, we used an advanced modeling construct for dynamic population based on static information: recall the asterisk in Figure 13. We have the concept Person and may add the concept Birthdate, both corresponding to clear real world counterparts that essentially don't change. The Age concept may now be dynamically populated based on a calculation function upon Birthdate and it will be repopulated every time it is accessed. When this information system is queried, a Person's Age can be populated and everything is fine. Of course, this is a fairly simple example of time dependency. But the idea remains the same for more difficult concepts. When time is the only dynamic part, the asterisk role construct remains applicable and the added complexity will be represented in other roles and the calculation functions defining the population of the dynamic role.

3.2.2.2 Interpretation dependency: Walking Distance

When defining concepts and relations, we are naturally searching for a strict *definition*. Sometimes, such a definition cannot be given since it is subject to interpretation. In literature, this kind of concept is called a *linguistic variable*, a concept originating from the area of knowledge representation in fuzzy logic [8]. An example of this is the concept "Walking Distance". Certainly, some meaning can be given to this concept. Most typically, it will be a distance that is "not too long" and it intuitively carries a notion of time involved to travel this distance. Often the concept of "Walking Distance" is used when asserting a public transportation route, or determining whether a location can be reached for someone that doesn't have any other means of transportation. What is considered walking distance may vary a lot between elderly and young people, but also between seemingly comparable people. However, this variance adheres to underlying statistical rules. Therefore, we use probability theory to approach this kind of dynamic information, more specifically the expected value distribution graph. Let's look at how the expectation about what is considered walking distance may be represented as a graph.

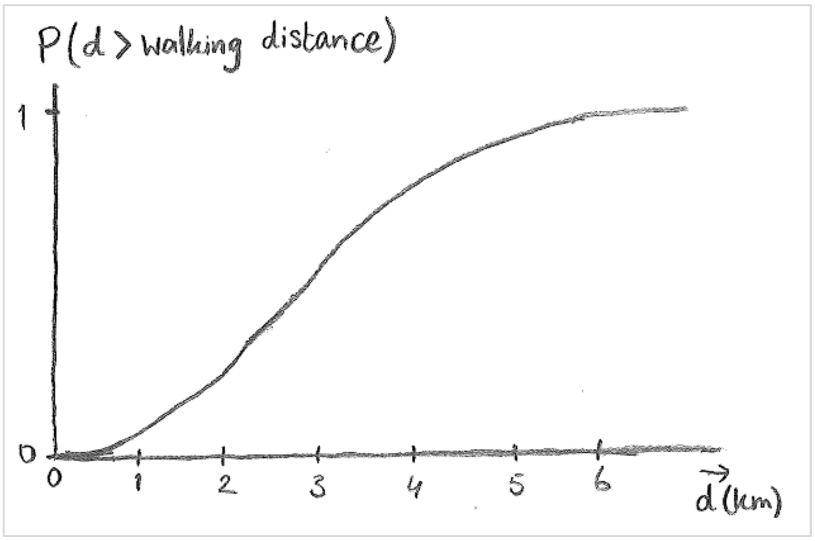

Figure 18 – Walking distance as a linguistic variable

On the horizontal axis, we simply put the distance. We are talking about walking *distance*, after all. The vertical axis hosts some probability, of which the values are fixed between 0 and 1. In this case, we are interested in the probability that given distance is considered *beyond walking distance*. Note that we achieve the exact opposite graph when plotting the probability "distance is within walking distance". We must keep in mind that the plotted graph describes the expected distribution for all Clients within a certain statistic group, such as "all Dutch people". When a random Dutch Client walks into the counselor's office, this graph should describe what can be expected of his interpretation of the concept "walking distance". By using this method, we create a statistical expectation of a semantically indefinable concept, which acts as a definition. To show how this works, let us return to the Alice example we saw before. One of the conditions of Gym B was that the Client should be living "within 20 km from Gym B". This condition is seemingly discrete: Alice either lives within 20 km or she does not, 1 or 0. However, in Alice's mind this distance condition may be less discrete. It might be one of these three:

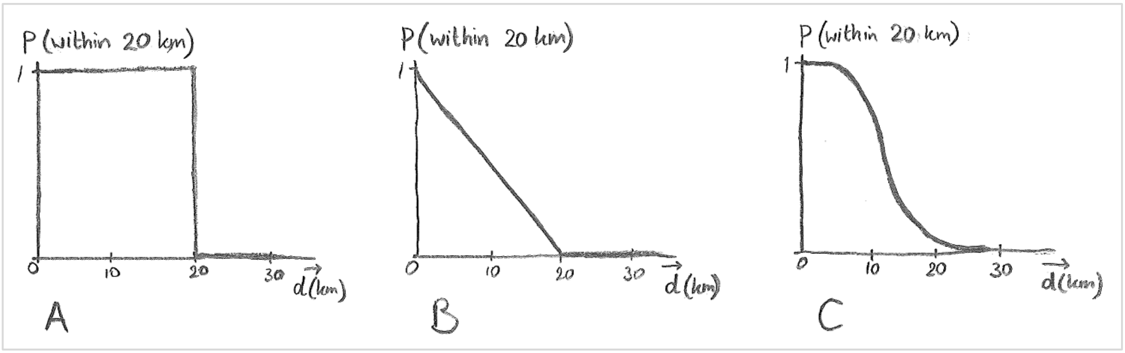

Figure 19 – Different interpretations of "within 20 km"

As we see, the concept "within 20 km" is interpreted in three different ways. Interpretation A follows the standard definition: you are either within 20 km (1) or not (0). In interpretation B, there is a linear reduction from 1 to 0 while the distance gets higher. The farther away you are, the less you interpret your situation as being "within 20 km". After 20 km, interpretation A is equal to B. The borderline of twenty kilometers is equally strict. The last interpretation releases this strictness: here we follow roughly the same pattern as B, where the feeling of being "within 20 km" will decline even faster in interpretation C. However, the borderline is not at twenty but at thirty kilometers, extending the limit because of the notion that it doesn't really matter that much whether we would have to travel 21 km or 20, after all. Interpretation C comes near the condition "should preferably be close, but since 20 km is quite far away, it doesn't matter when it's not exactly within 20 km".

This approach of linguistic variables enables us to express information that is subject to interpretation, and use this information in the knowledge base and querying. The probability value can directly be used to determine relevancy of a given Solution.
When used for the *knowledge base*, it enables us to store a Condition. In this case, there might be three gyms offering services in a radius of "twenty kilometers", but when investigated more closely their service area condition looks like graph A, B and C, respectively. Now we can see that Gym A will always be applicable for a client who lives within 20 km, while B is more suitable for clients living closer than 15 km and C for clients living farther than 15 km. Furthermore, C is the only one which services clients that are not strictly within 20 km.

When we apply the idea of linguistic variables to the *query* side, we view it as the searcher's interpretation. This is most useful when the knowledge base does not provide a definition, but just the variable itself. It might be left to the searching client to think about whether a gym is within walking distance *for him*. The gym itself would just state that a condition is that you live "within walking distance". For one client, this may mean that every gym within 5 km is equally suitable (according to interpretation A) while another client may strongly prefer something very close and have a larger maximum distance (according to interpretation C). Being able to handle this kind of variables at both sides of the information retrieval process makes a huge difference.

3.2.2.3 Environment dependency: Conditions and SolvableProblems

The Condition concept is already subject to formulation and validation challenges when viewed as a static concept, as we saw during the construction of our model. However, there is also a dynamic dimension involved. Conditions may only be satisfied under certain circumstances, that may hold at one moment and be negated at another moment. This effect notably occurs in our model when a Problem is splitted into different SolvableProblems. Conditions for each SolvableProblem are stored individually and may be validated as such, but this validation occurs *at counseling time* in our model. Some conditions depending on another client environmental situation may be satisfied at that time, but not at another time, and vice versa. For example, suppose Gym A in Alice's example would have explicitly required her to have attended the wheelchair course of Gym C. Alice, not having attended this course at the time, might therefore not realize that Gym A has an interesting offer for her. Failing to satisfy the condition associated to playing youth basketball in Gym A at counseling time, she might have had the choice between a wheelchair training and playing hockey.

When analyzing what the best way to split up a Problem is, the changes occurring to the environment of the Client should ideally also be taken into account when validating the conditions associated with possible solutions. This is an even harder challenge, but handling it to some extent will reduce the number of false positive or false negative results.

4 Towards a Semantic Web application

When implementing a traditional database driven application, a conceptual model like the one we constructed in the previous chapter may to be transformed into a relational database. Because this transformation may be performed in different ways, we put it inside the interpretation layer (see Figure 4). With the resulting database, knowledge data may be structurally stored and retrieved. However, useful data activities like interpretation and interconnection largely depend upon customized human action.

In a mature Semantic Web, this situation is improved in two ways. First of all, data is annotated with *semantics* that are both machine and human interpretable. Because of these available semantics, reasoning rules are now on the surface and search results or answers to questions may be based upon them rather than on some invisible strategy. Therefore, the expected reasoning and understanding level which can be reached by applications will be more reliable. Instead of providing users with just a result that hopefully looks like what they needed without really expressing why, conclusions may now be supported by arguments derived from semantics. Secondly, the data structure is connected to other (semantic) data structures, thus forming a global *web* of data. Because the Semantic Web is like a worldwide database, the combination of world facts to draw conclusions from is no longer limited by any boundaries, as long as there is some connection between different sources.

Both improvements are required to increase the functionality offered by a typical current web information source. When there is a lack of semantics, conclusions are not supported by strong arguments and as a result they will often be too untrustworthy to be of use. When the web is too small, combinatory power is too limited and few interesting facts may be derived. For both improvements, another supporting data structure is needed. In other words, we should transform our ORM/ORC model to something else than simply towards the traditional relational database.

In this chapter, we will investigate the Semantic Web more closely. We start off in section 4.1 with an overview of W3C standards defining the Semantic Web languages. After that, we will use section 4.2 to investigate research efforts towards transforming non-semantic structures into their semantic counterparts. In section 4.3, we conclude with an analysis of how to proceed from our current situation.

4.1 Semantic Web languages

When investigating Semantic Web languages, it's important to distinguish between the conceptual and implementation aspect. Since we are dealing with *web* languages, they are technically founded upon XML. This metalanguage is commonly used as a mediating language between different applications. In section 4.1.1, we will first look at this XML foundation to place the Semantic Web languages into a Web context. Consequently, we will look at the main conceptual language standards found in the Semantic Web. As we will see, they are RDF and RDF Schema (4.1.2) and OWL (4.1.3) and SPARQL (4.1.4).

4.1.1 XML: the mediating metalanguage

Web pages are mostly written in HTML. Since the first webpage was written, most significant improvements have been in the area of *what* is written and *how* it is written. While creating a Web page may still be done in the old-fashioned way, by hard-coding every single line in a text editor, nowadays there is a variety of tools to make this process more easy. Advanced text editors provide us with syntax highlighting and assist in writing by showing available language options. Moreover, there are tools at a higher level of abstraction which let users design a Web site without writing code themselves. This has enabled a very large group of users to create their own Web page. But the underlying foundation essentially remained good old HTML, designed to present a document in an easy way - for humans. Markup is targeted at the presentation level, not at the content level. When we want to use content automatically, a machine should be able to consistently and predictably retrieve bits of information from the document. When this information is not clearly structured, information retrieval is subject to errors or at least less predictable accuracy. To cope with this, eXtended Markup Language (XML) was designed.

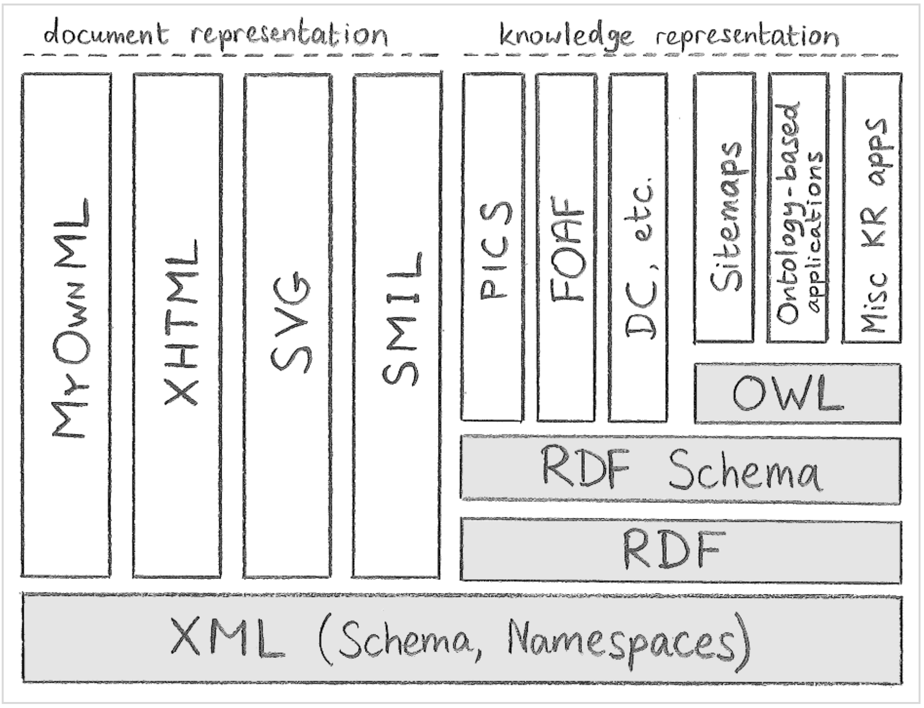

Figure 20 – Document and knowledge representation languages on the Web (adapted from **[9]**)

XML is based on the same general ancestor as HTML: the Standardized General Markup Language (SGML). Therefore, it also uses tags to enclose relevant parts of data. However, the meaning of these tags is not defined. While in HTML every tag has some fixed meaning and suggested usage, this is left open in XML. There are only rules which define valid usage of tags in general, such as the obligatory closure of every tag and nesting rules. It is up to the programmer to define a suitable vocabulary and process given data. These vocabularies may be defined, so one does not just adhere to XML but for example to XHTML. Strictly spoken, an XHTML document is "just an XML document", but because all used tags happen to be XHTML tags, a browser knows how to display all content as intended by the programmer. Using XML as an underlying metalanguage and some predefined vocabularies on top of it has the advantage of extensibility ("everything may be expressed in XML") without losing interpretability ("we know what things from our vocabulary mean").

Figure 20 provides an overview in which XML is the foundation of all document and knowledge representation. XHTML, the XML-based Web language, is just one of the available predefined vocabularies. In this figure, we see mostly language standards recommended by the the World Wide Web consortium (W3C[9]), which develops and maintains proposed global document standards. But since XML is in fact the underlying general markup language, it is used for a wide array of specialized vocabularies in small domains. Because machines can parse XML everywhere and interpretation rules are neatly structured by the vocabulary definition, "talking XML" quickly became the way to go. In this XML-based communication, there is a clear distinction between (structured) data and presentation. The structure may be enforced by XML schema rules; these same rules may be used to know what to expect. However, this information is still subject to human interpretation. If we see that there is a list of Books, which contain Authors and PublishDates, we consider this as something else as a list of Animals, which contain BiologicalNames and AverageAges. For a machine, both lists appear identical. Also, common structuring concepts such as classifying and hierarchy, which help humans (and machines, for that matter) to navigate through a lot of information, have no special predefined meaning in XML. If we want to interpret data with a machine, all interpretation guidelines should be defined again for every XML application, unless both adhere to the same standard and the guidelines may be defined for this standard.

When the amount of (XML represented) data on the Web grew, the need for more intelligent interpretation and reasoning became apparent. In fact, when we search for information, we most often seek *knowledge* instead of a *document*. We don't really care how the information is presented or where, but we just need the information. Because of the enormous size of knowledge available on the Web, represented in even more documents, it becomes more and more of a problem to manually sift through all potentially relevant documents, searching for bits of knowledge that provide us with an answer when combined in a logical way. When all documents are represented apart from each other in their own (human-interpretable) vocabulary, help of a search engine remains limited to providing us with a set of documents that seems as relevant as possible. And it's up to us to interpret and combine the pieces of information.

The Semantic Web languages help to enrich the current Web with semantical annotations, so the knowledge aspect may be stored in a standardized way. This complementing relation between the two "versions" of the Web has been topic of various research efforts. Ankolekar et al. sum it up nicely as: "future web applications

[9] http://www.w3.org

will retain the Web 2.0 focus on community and usability, while drawing on Semantic Web infrastructure to facilitate mashup-like information sharing" [3]. Iyad Rahwan also pointed out the benefit of using the human element from Web 2.0 in the Semantic Web, especially when trying to structure subjective natural language [10]. The community aspect of Web 2.0 is of such great value because extracting knowledge and intelligence aspects automatically from natural language text is most difficult. Therefore, human effort is vital for constructing desired applications. Semantic Web languages are basically only providing a framework to store the results of this human effort in a machine-usable way.

In Figure 20, the Semantic Web languages are colored grey. Unsurprisingly, they are all technically an XML implementation. We would like to pay attention to the depicted knowledge representation standards based on RDF: Platform for Internet Content Selection (PICS), Friend-Of-A-Friend (FOAF) [11] and Dublin Core (DC) [12]. PICS provides a standard way to describe a rating of online resources, the FOAF standard contains concepts and relations typically used in a social network context. The Dublin Core standard defines general markup metadata to describe properties of resources, like author, title and publication date. We will use some of these constructs where applicable in our examples.

We will now take a closer look to these essential Semantic Web languages and their use. Note that RDF and RDF-S are depicted on a different level than OWL. This accounts for a difference in usage: RDF is used to describe and structure the *actual* knowledge that's available, while OWL is used to describe what *kind* of knowledge is available and what (logical) rules apply when looking at the structure. In other words, OWL describes knowledge at a higher level of abstraction. Because of this distinction, we will first look at RDF and RDF-S and subsequently at OWL. We used the second version of "A Semantic Web Primer" by Antoniou et al. [13] as a reference source. Furthermore, all latest details may be found at their respective W3C web pages, which are kept up to date with latest developments.

4.1.2 RDF and RDF Schema

RDF[10] (Resource Description Framework) is essentially a data model. Although the Semantic Web is centered around the XML representation because of its machine usability, we should keep in mind that other representations are also possible. For example, a graph representation is more suitable for humans. To gain an understanding of RDF, we will therefore start with this representation and keep in mind that it can be easily translated to a version that is interpretable for machines.

4.1.2.1 Data model

The main component of any RDF model is an object-attribute-value triple, also called a *statement. Objects*, or resources, are things from the domain we want to be accessible. They are identified by a Universal Resource Identifier (URI). What a URI looks like

[10] http://www.w3.org/RDF/

depends on the kind of object it identifies. For example, a web page may be identified by its URL, and a book may be identified by its ISBN number. In Figure 21 we see an example of some objects, represented in the graph form.

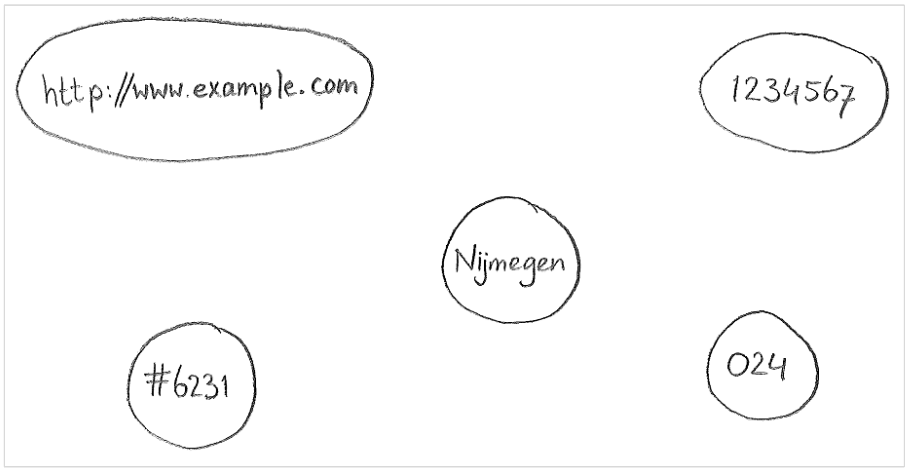

Figure 21 – Objects

When looking at this piece of RDF, humans may have an idea of the meaning of some objects, but in fact they are just arbitrary instances of which the class and type are unknown. The object "#6231" is arguably the most semantically unidentifiable of the lot. Keep in mind that the values inside objects are short-hand notations of URIs. We may look upon them as unique identifiers of instances found in the real world domain. In fact, every object represented in the graph corresponds to a unique object somewhere in the real world. Denoting objects is nice for a start, but they become semantically interesting when they are really different, not only by name. For a machine without any notion of what words or concepts mean, an object is only given unique meaning by its relations with other objects.

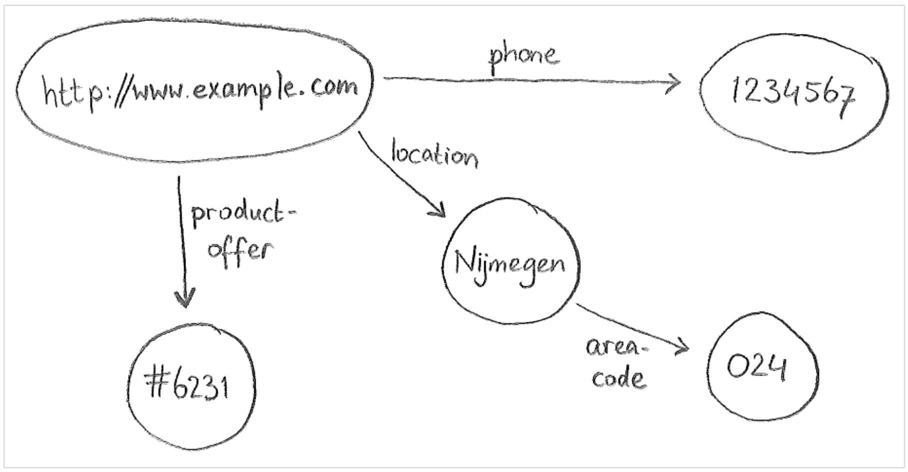

Figure 22 – A couple of object-attribute-value triples

Object types which have the same relations attached to them, are treated as semantically equivalent. To assign semantics to a given object, we therefore have to be able to describe its unique *attributes* or properties. In RDF, properties are identified by a URI just like objects. They are just a special kind of resource: the attribute itself is also possibly an interesting "thing" to describe further. To describe a domain, we write down statements asserting resources and their attributes. Such statements have the form of a triple; a predicate and two associated variables. Either part of this triple may be a URI-identified resource or a plain string text. An example of a statement triple is "Jos Claessens is the writer of thesis 612". Logically, this might also be represented as *WriterOf (Jos Claessens, Thesis 612)*, which shows more clearly that it's a binary predicate. For the human reader, this single statement already carries some intuitive meaning, because we have an understanding of writing a thesis, and we may recognize person naming. Besides its general meaning, this statement may be ambiguous. It depends on how many "Jos Claessens" you know, how familiar you are with the concept of writing and how many kinds of thesis numbering systems use the format used by the Radboud University. To minimize ambiguity, we should provide a URI for all parts of the statement that may be ambiguous, in this case: all. It might better be reformulated as "<Person with Dutch ID number 1234.56.789>, <writer-of>, <Thesis with number 612 according to Radboud University Computer Science numbering>". The semantics of all these parts may be formally described somewhere. In RDF, all statements have the form of such triples, which are equivalent to binary predicates. It is the most elementary way of expressing relationships between two entities. ORM also enforces that statements fulfill certain requirements when it comes to the number of used roles. The number of roles is not limited to two per fact type, but due to the *n-1 rule* the number of roles in a fact type may not surpass the number of roles of its largest uniqueness constraint by more than 1. Relations between entities are expressed in *n-ary* predicates, with $n \geq 2$. When converting an ORM model to a Semantic Web implementation, n-ary relations will have to be transformed to a set of binary relations. We will use this technique in our own transformation method in chapter 5.

RDF supports statements about statements (reification). This concept is identical to the ORM notion of objectification. Having only binary predicates feels limiting here, because for a simple pointer to a RDF statement already three binary predicates are needed. This is not a problem for expressive power, because we can use the same transformation method as with n-ary predicates, but nevertheless it is rather cumbersome.

RDF Schema

RDF is just about describing resources and their relations, in other words it is about the *instance* part of a model. However, in practice we want to express more general statements. For this, we need RDF Schema, or RDFS[11]. The main addition to RDF is the use of *classes*.

[11] http://www.w3.org/TR/rdf-schema/

We may express the following things:

- class definitions
- property definitions (domain and range)
- class and property hierarchies

In RDFS, we may construct a structure very similar to a domain model; RDF instances may be seen as the population of this model. They are given a type from the associated RDF Schema. For example, we may define the property "Student writes thesis ThesisNumber". If "Jos Claessens" is a Student, he is in the correct domain. Similarly, "612" should be a correct ThesisNumber. Classes and properties may have subclasses and subproperties. In that case, children inherit the properties associated with their parents. For example, if any Person might write a thesis, and Student is a subclass of Person, then it will still be enough if "Jos Claessens" is defined as a Student. Because a Student is also a Person, he is inside the domain. Contrarily, if only Students may write a thesis and Jos is defined only as a Person, he may not write a thesis. Or at least, that is not a valid statement in the RDF schema.

Besides these basic functionality constructs, RDFS also includes some utility properties to further enrich class definitions. These are not strictly necessary, but they may save some time when working with (especially large) RDFS documents. There are two main purposes of utility functions:

- Stating where a (more complete) definition of a resource may be found. This is especially useful when a resource is defined at multiple places across the Web.
- Attaching a human-readable comment or label in unstructured natural language, to improve human understanding of RDFS classes and their purpose.

4.1.2.2 XML representation

For completeness, we will briefly take a look at the most important RDF(S) constructs in XML with an example. More elaborated examples may be found in [13].

<u>Namespace definition (XML)</u>

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:ex="http://www.example.com/ex-ns#">
...
</rdf:RDF>
```

Any RDF document should start off with some namespace declarations. In addition to disambiguation, which is the main purpose of using namespaces in XML, this specifies the semantics of used constructs. When a RDF knowledge entity has been defined somewhere, it may easily be reused by including the original definition as a namespace. Our new definitions file is thereby connected to the larger graph of interconnected RDF documents. Or in other words, it is connected to the Semantic Web. Note that the "ex" namespace is our own virtual example namespace

Class definition (RDFS)

```
<rdfs:Class rdf:ID="supervisor">
  <rdfs:comment>In this case, we mean thesis supervisor</rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="student" />
<rdfs:Class rdf:ID="thesis" />
```

We defined the (thesis) supervisor class, the student class and the thesis class.

Subclass definition (RDFS)

```
<rdfs:Class rdf:ID="professor">
  <rdfs:subClassOf rdf:resource="#supervisor" />
</rdfs:Class>
<rdfs:Class rdf:ID="manager">
  <rdfs:subClassOf rdf:resource="#supervisor" />
</rdfs:Class>
```

Both professors and (business) managers may be a thesis supervisor.

Property definition (RDF / RDFS)

```
<rdf:Property rdf:ID="writes">
  <rdfs:domain rdf:resource="#student" />
  <rdfs:range rdf:resource="#thesis" />
</rdf:Property>

<rdf:Property rdf:ID="supervises">
  <rdfs:domain rdf:resource="#supervisor" />
  <rdfs:range rdf:resource="#thesis" />
</rdf:Property>

<rdf:Property rdf:ID="title">
  <rdfs:domain rdf:resource="#thesis" />
  <rdfs:range rdf:resource="&rdf;Literal" />
</rdf:Property>
```

We define three properties: the first two are writing and supervising a thesis. The third is the title of a thesis. Note that its range is the RDF Literal, which is essentially just a string. After defining relevant classes and properties, all that remains is providing a sensible population.

Instances (RDF)

```
<ex:thesis rdf:id=”612”>
  <ex:title>
    Deploying the Semantic Web in the Welfare Environment
  </ex:title>
</ex:thesis>

<ex:student rdf:id=”JC”>
  <foaf:name>Jos Claessens</foaf:name>
  <ex:writes rdf:resource=”#612” />
</ex:student>

<ex:professor rdf:id=”TvdW”>
  <foaf:name>Theo van der Weide</foaf:name>
  <ex:supervises rdf:resource=”#612” />
</ex:professor>

<ex:manager rdf:id=”JWS”>
  <foaf:name>Jan Willem Schoenmakers</foaf:name>
  <ex:supervises rdf:resource=”#612” />
</ex:manager>
```

In this example, we can clearly see how verbose the XML representation is compared to the graph representation. This accounts for the fact that it is targeted at machine interpretation rather than human interpretation.

4.1.3 OWL

RDF and RDF Schema allow us to create a simple knowledge base, as we saw in the previous section. But their expressiveness is deliberately limited. Consequently, we need another language for those cases where more expressiveness is needed. OWL[12] is such a language, positioned on top of RDF/RDFS in Figure 20. It is the W3C recommended Web Ontology Language, originating from its predecessor DAML+OIL which was on its turn the result of joining the American initiative DAML-ONT and the European initiative OIL. In this section, we will look at the addition this language brings to RDF/RDFS.

Compared to RDF/RDFS, OWL mainly adds features to further refine class definitions and restrictions. Definition was limited to placing new classes into a hierarchy and hinting towards related classes with `rdfs:seeAlso`. Restrictions were basically only possible with the global scope (domain and range) introduced in RDFS. In OWL, the following features are added:

Definition

- Classes may be defined based on already existing other classes, using well-known set operators. They are also called Boolean combinations. We are talking about one class being `owl:disjointWith`, `owl:complementOf`, `owl:equivalentClass`, `owl:unionOf` or `owl:intersectionOf` another (set of) class(es). These properties may be nested as desired.

[12] http://www.w3.org/TR/owl-features/

- With the `owl:oneOf` construct, all possible elements of a class may be enumerated as its definition.
- RDF Properties are specialized into two kinds of properties in OWL: object properties, which relate objects to other objects, and data type properties, relating objects to a data type value. If applicable, properties may be defined as the `owl:inverseOf` or `owl:equivalentProperty` of another property.
- Property properties may be expressed, for reasoning and verification purposes. Available constructs include `owl:TransitiveProperty`, `owl:SymmetricProperty` and `owl:FunctionalProperty`. They correspond to mathematical properties with the same name. Furthermore, we have an `owl:InverseFunctionalProperty`, which indicates that two objects with that property may not have the same value. Transitivity and symmetry may only be associated with object properties, while functionality may be associated to any property.

<u>Restriction</u>
Restrictions are encapsulated in an owl:Restriction element and may take various forms.

- Scope of properties may be restricted in a more finegrained way, by using `owl:hasValue`, `owl:allValuesFrom` or `owl:someValuesFrom`. With these constructs, we may define a relation between a concrete resource and a class of other resources or express universal quantifications or existential quantifications, respectively. For example, we might specify that "JC" must be the writer of thesis "612", or that in case of a computer science thesis, both the student and the supervisor should be from that department.
- Cardinality of classes may be expressed by `owl:cardinality`. In case of a cardinality range, we may use `owl:minCardinality` and `owl:maxCardinality`. We might for instance specify that a thesis should have at least two supervisors.

This list is not complete, but it sums up the most important features added by OWL. For a complete overview, we refer to the official W3C page.

The main purpose of OWL is to support reasoning about the constructed ontology. Reasoning support is useful for checking ontology consistency and finding new relations between classes which may be unintended. Besides this reasoning on the model level, we may also use reasoning on the instance level, to classify instances in classes and for query answering. To be able to use automatic reasoning, we need a well-defined syntax (so the language is machine-readable), a well-defined semantics (so the language is unambiguously interpretable) and a good balance between needed expressiveness and computational efficiency. OWL meets the syntax requirement by adhering to XML, which is perfectly machine readable. Formal semantics are usually provided by logical formalisms to which OWL may be mapped. Reasoners which are able to use this logical formalism may consequently also reason about OWL documents. Our focus is upon the last requirement: a good balance between expressiveness and computational efficiency.

Perhaps unsurprisingly, the full set of OWL and RDF/RDFS constructs (conveniently named OWL Full) may lead to situations of computational intractability. In any application where reasoning should be reasonably efficient, this possibility is undesirable. Therefore, work has been done to specify the subset of OWL constructs that is still computationally efficient. For this, the largest part of first order logic for which efficient reasoning is still possible, known as Description Logics, was used as a reference. The result is OWL DL, in which the following constraints must be obeyed:

1. Resources may only be one type of object as found in the OWL vocabulary. For example, a class may not also be an individual. This typing should be *explicit*. For example, when *C1* is an `owl:subClassOf` *C2*, both *C1* and *C2* should explicitly be defined to be an `owl:Class`.
2. Because a property may not be both an object property and a data type property because of rule 1, property properties that were applicable to both kinds of properties are now only applicable to object properties.
3. Cardinality restrictions may not be placed upon transitive properties.

The third official W3C subset of OWL is OWL Lite, which we will not further discuss here. It's most important to realize that this balance between expressivity and computational tractability is a real concern and that we should try to find the subset of OWL that is just expressive enough to reach our goals without giving up too much computational efficiency. The choice which subset this is, depends too much on the situation. However, we may conclude here that OWL DL is a subset with an arguably well overall applicability.

OWL XML syntax is an extension upon RDF syntax. In fact, it inherits all constructs we saw earlier; only the `rdf:Class` and `rdf:Property` have been replaced by their OWL counterparts for computational purposes. Elaborated examples may be found in [13].

4.1.4 SPARQL

SPARQL[13] is the W3C recommended query language to use on semantic data structures. It enables powerful reasoning questions, structurally formulated to suit the RDF representation. SPARQL consists of three different parts: the *query language specification,* a *query results* XML format and *data access protocols*.

Syntactically, the query language closely resembles SQL. Although the underlying data model is fundamentally different, it is only minimally reflected in the language constructs. This makes migration from a relational database situation to a semantic web situation easier. Just like in SQL, a typical SPARQL query has the form `SELECT...WHERE...`. In this expression, what is searched for appears after the `SELECT` keyword. In case of SQL, this is usually a set of column names selected `FROM` a certain table. Because we choose actual column names, a `SELECT...FROM...` query is enough to return a subset of the queried table. In SPARQL, there are no column names, so as a result they cannot be referenced. Instead, we can define variables that will be

[13] http://www.w3.org/TR/rdf-sparql-query/

given a meaning by the `WHERE` clause. In SQL, this clause is used to restrict the results. The `WHERE` clause acts as a filter upon the selected results. In SPARQL, it acts as a definition of the variables. Restrictions may directly be included. We might select *x* and *y* where *x* foaf:knows *y*, but also include that *y* must be a professor, or that *y* should have at least 3 friends. If desirable, local variables may also be introduced inside the `WHERE` clause. In this example, we could only select *x* with the same relatedness to *y*, which would still be expressed inside the `WHERE` clause. To restrict the accepted return values, a SPARQL `WHERE` clause may contain the keyword `FILTER`. Using this construct, results are filtered based on a regular or arithmetic expression.

Designing a suitable query engine has been topic of various research efforts, for example towards the Semantic Web Search Engine (SWSE) [14] and Swoogle [15].

4.2 Transforming to the Semantic Web

Defining Semantic Web languages is a first step towards the Semantic Web. However, to actually construct the Web itself, documents should be written using these languages. Most of the time, they will not be written from scratch but rather transformed from existing structures or integrated in existing languages. Depending on the nature of the origin of a future semantic document, different strategies may prove to be best here.

Existing HTML structures may be targeted at human readers, but this usually means there is some structure in them. This structure may be exploited to design (ad hoc) transformations extracting a semantic document from arbitrary HTML tags. Web content creating tools may be enhanced to have users publish their new pages enriched with semantic metadata. Since enriching current web content seems like an easier process than creating full RDF documents, there have been several initiatives to support this intermediate solution. Using RDFa[14], a W3C recommended extension upon XHTML, RDF may be embedded in existing XHTML pages using special tag properties.

In this section, we investigate the possibilities to transform the different aspects of the Social Map to the Semantic Web standard languages. We will look at the transformation of three relevant types of conceptual structure: an ORM model, a thesaurus and a relational database.

4.2.1 Transforming an ORM model

Transforming an ORM model towards OWL enables using ORM for ontology engineering. The benefits of ORM for usage by domain experts (which is why we used it to create our model in the first place) would then also hold for ontology engineering. Logical sentences, which will ultimately be represented in a machine-accessible language, may first be expressed in structured natural language, which may easily be verified by human domain experts.

As we saw when we discussed OWL, the focus is upon the balance between expressivity and computational tractability. Description Logics provide a logical

[14] http://www.w3.org/TR/rdfa-syntax/

foundation which is targeted at maximum expressivity while still being decidable. Therefore, it has extensive automatic reasoner support and an associated OWL language subset (OWL DL). Therefore, mapping ORM into Description Logics has been topic of various research efforts. The most notable results have been achieved by Mustafa Jarrar [16] and C. Maria Keet [17]. In their research, they both mapped ORM into $\mathit{DLR}_{\mathit{ifd}}$, which is asserted to be "one of the most expressive description logics" [16]. Jarrar showed that 27 of 29 ORM constructs and constraints are mappable and therefore decidable. Consequently, there are two undecidable constructs: the acyclic ring constraint and a frequency constraint spanning more than one role. Since we don't make use of these two in our ORM model, this might lead us to conclude that our ORM model does not exceed complexity of description logics. Unfortunately, we should refine this quick judgement.
First of all, we don't make use of classic ORM, as was mapped by Jarrar. Neither do we make use of ORM2, as was mapped by Keet. We have some additional constructs, specifically generalization and power type, that allow for greater expressivity. But therefore, they are also possibly introducing greater complexity. The lack of support for these constructs is logical since the mappings by Jarrar and Keet do not intend to include them, but nevertheless such mapping should be available before we can transform our model.
Although we did not expect coverage of our constructs, we did expect the regular mapping to be satisfactory. Unfortunately, some core constructs of base ORM were not mapped completely. Keet already found some incomplete and even incorrect mappings in Jarrar's work. Firstly, he did not always map ORM to $\mathrm{DLR}_{\mathrm{ifd}}$, but sometimes borrowed from other types of DL languages. Secondly, some mappings are also incomplete or even incorrect [17]. Because of this investigation, less constructs are mappable than originally specified by Jarrar. Keet defines ORM^{-} as the subset of ORM that is mappable, according to her transformation rules. This is a rather arbitrary definition that's still inaccurate in our opinion. Both Keet and Jarrar neglect the transformation of subtype defining rules, which admittedly form the hardest and least specified part of the subtype construct. However, without such transformation, we can hardly speak of a successful mapping of the subtype construct. In fact, the approach towards subtyping used by Keet reflects the generalization construct rather than the subtype construct. Furthermore, both Jarrar and Keet neglect an essential external uniqueness constraint when interpreting the objectification construct, as we will see later in this section. Therefore, their mapping towards $\mathrm{DLR}_{\mathrm{ifd}}$ is incomplete.
Finally, as was also concluded by Keet [17], mapping ORM to description logics in general may be feasible, but not to any single language. Most DL languages are capable of handling certain constructs well, but the mapping of other constructs is unsatisfactory. Therefore, current results about ORM mapping towards DL mainly serve as an indication of general ORM construct decidability, rather than being useful for transformation of (complex) models towards a real Semantic Web application. This target is even further away because the standard language OWL DL does not make use of $\mathit{DLR}_{\mathit{ifd}}$, but *SHOIN*. The main reason that OWL DL uses a different DL language is a performance concern. *SHOIN* was designed to be "a compromise between expressive

power and decidability" [18]. Here, the goal is to maximize *practical* decidability, i.e. having maximum expressive power without losing too much performance. This concern is most critical when using logics on the web, for example in a query language. Compared to *DLR*$_{ifd}$, some expressivity is sacrificed to gain performance. Since it is the official underpinning of our target model language, it makes sense to look more closely at the difference in mapping possibilities. Jarrar already followed up his research of the general level of ORM decidability and investigated *SHOIN*/OWL more closely [18]. The difference clearly shows: when mapping into *SHOIN*/OWL, only 22 of 29 ORM constructs could be transformed. When we take into account the criticism we discussed earlier, this result should be regarded to be at the optimistic side. To aid in the ORM schema validation and transformation process, all mapping rules have been implemented as an extension to the DogmaModeler tool. This tool allows (visually) creating an ORM diagram, mapping it to the DL interface DIG and reasoning about the resulting description logics. At the moment, three types of reasoning services are implemented (schema satisfiability, concept satisfiability and role satisfiability). These validate the ORM model itself using logics. Extended reasoning services like constraint implications, inferencing and subsumption, as well as support for OWL syntax export are planned for the future, but as for now they have not yet been realized. Note that this has been the case for the last two years since Jarrar published the mappings and related implementation.

Unfortunately, current state of the art research targeted at mapping ORM to Semantic Web languages has not been able to provide us with a fully functional and satisfactory transformation method. Some crucial constructs are missing or incomplete, and the logic languages and reasoners currently supporting the Semantic Web are still in development and should not be expected to be a full match for ORM in the near future. This is a bit of a disappointment, but it may be explained by looking at Figure 4 once more. In this picture, description logics languages should be put towards layer 4. This is because the languages have syntax and semantics that are targeted at a specific application field. Recall that the ORM model is in layer 2. We believe that not all ORM constructs can simply be transformed towards the same description logics language because of the gap between layer 2 and 4, which is simply too large. Therefore, we prefer to construct a mapping towards a more abstract, "level 3" kind of logics, in which all ORM constructs may still be expressed. From there, transforming various parts to suitable description logics should be easier and more straightforward. We will provide such transformation method in chapter 5.

4.2.2 Transforming a thesaurus

A thesaurus is a special kind of conceptual structure. Concepts in a thesaurus are structured based on their relatedness, without investigating why this relatedness exists. Typical relations are *broader term* and *narrower term,* structuring based on generality of concept. Also, one may navigate to similar terms, which are considered a synonym in isolation, but differ in their environment. By looking at the relations of a term, one may distinguish between homonyms as well. In information retrieval, a thesaurus is used for indexing and tagging documents. Since most search methods are word-based, it's valuable to know for a given word how related it is to the exact query word. Besides during the initial retrieval process, a thesaurus may also be used to refine search results afterwards. Terms that appear in the query itself or in highly ranked results may indicate that the user is interested in related terms as well. They may be shown together with actual query results, to guide the searcher when actual results are not satisfying. In fact, the current welfare environment information system uses this strategy already. However, the representation of the underlying thesaurus is technically structured in a legacy format, a document standard specifically developed for thesauri. It conforms to the ISO maintained thesaurus standard ISO 2788:1986. Interestingly, this standard does not account for the used representation, but rather for available relations between terms and their (short-hand) notation. Because the available semantic relations are standardized, the conversion from any representation to Semantic Web language should be feasible, as long as these relations are reflected. Therefore, work has been done to define a neat semantic web standard for thesauri, called the Simple Knowledge Organization System (SKOS). Since June 15, 2009, the SKOS specification is a W3C proposed recommendation. According to W3C's technical reports website[15], "a Proposed is a mature technical report that, after wide review for technical soundness and implementability, W3C has sent to the W3C Advisory Committee for final endorsement". This means SKOS is not yet an official recommendation, but it will expectedly become one in the near future. Therefore, it has already led to some interesting related research. For example, Van Assem et al. have shown that it is possible to automatically convert more traditional thesauri to SKOS, as long as they comply to the ISO 2788 standard [19]. Their method values interoperability higher than completeness: although some thesaurus relations might not be (fully) transformed to SKOS, there is a clear interoperability advantage of using a semantic web standard of which the semantics are known and predictable when the thesaurus is used in a larger semantic web application. The method of Van Assem et al. consists of analyzing thesaurus content, creating mappings to SKOS constructs and consequently writing a conversion program that converts the thesaurus to a SKOS ontology based on these mappings. Case studies showed that the whole process typically takes one analyst around two weeks to complete. They also showed that actual thesauri are often less neatly standardized than one might expect, which might reduce the applicability of the transformation method.

4.2.3 Transforming a relational database

Relational databases are a well-known existing structure supporting many current web-based applications. There are several approaches to the reuse of such a conventional database. We will investigate them briefly.
Motik et al. defined concepts from relational databases in OWL terminology, to bridge a terminology and concept gap they identified [20]. This gap is caused by the logical

[15] http://www.w3.org/TR/

approach to the "world". The relational database model uses the closed world assumption, while OWL uses the open world assumption. This affects reasoning about data that doesn't satisfy constraints. For example, when a certain property is explicitly required to be filled in a database model, but it isn't in a particular case, the relational model would consider this an error, while the semantic model would just consider it a critical, but as for now unknown property. Due to the open world assumption, nothing that is unknown is thereby also considered nonexistent. Motik et al. extended the description logics model to support integrity constraints with the same intuitive meaning as their relational database counterparts.
Bizer and Seaborne constructed D2RQ, a language to describe mappings between a relational database and a related semantic model [21]. This mapping may be used in the Jena framework [22]. Jena is a complete framework to create and maintain a native RDF store, but it does not provide for migration from a relational database situation. Using D2RQ, Semantic Web applications may easily use data stored in a conventional, non-RDF database, for it will be represented as a virtual RDF graph. This technique allows for easy reuse of legacy data, but usage of the Jena framework may be too heavy for smaller applications, introducing an unneccessary performance hit. For example, in a benchmark performed by Svihla and Jelinek [23], their own METAmorphoses and Sesame [24] generally produce query results faster. These approaches all require significant human effort to reuse existing data. But there are also attempts to process existing structures automatically and extract semantic data from them. Although this may not be as complete and accurate as the manual methods, it may be enough in many cases or at least be a nice starting point that is a lot more desirable than having to start from scratch. One of the leading methods is provided by Stojanovic et al. [25]. They developed a mapping and migration architecture that automatically generates semantic annotations from database content, under human supervision. Instead of designing the complete ontology, the engineer only has to resolve ambiguities or unclear situations which will occur during the transition. Similar projects have recently been carried out by Hu et al. [26] and Cullot et al. [27]. The method can generally be summarized as follows: database tables are converted to ontology concepts, while columns and integrity constraints are converted to concept properties. Hierarchic relationships already present in the relational database, such as a foreign key to another table with higher level information, are translated into making the concept of the "lower" table a subclass of the "higher" table. We illustrate this with an example adapted from [27]. The Student table may be related to a Person table with names of all people related to the school (including teachers). Based on the student's ID, the database application may retrieve his name from the Person table. This construction will result in the generation of a Student concept, which is a subclass of the Person concept. The Student concept will then automatically use the property 'name' from its parent concept. Note that data types need to be converted to XML schema data types. Whereas a column data type is inherently present in a relational database, it has to be explicitly specified in the ontology situation to avoid content with multiple types, which is allowed by default in RDF.

When we combine the results of database and thesaurus conversion, we have a reasonable starting point for a Semantic Web application, which is already connected to existing data and supports the same usage. It is therefore an alternative to creating a domain model from scratch. The semantic structure resulting from the transformation step is more easily extensible to reach new possibilities, especially when compared to the old situation. To illustrate this, we refer back to Figure 4. When domain and database models are converted to semantic models, they can be interconnected by hooking on to a global environment model. By doing this, all information inside will become interconnected and useable, albeit in the most primitive way.

When not following the route of creating an ORM model and deploying a Semantic Web application from there, we propose to first create a global ontology which can be semi-automatically generated from existing data sources. This ontology will not contain much new information, but mainly serves as a starting point. It can consequently be edited by domain experts from different parts of the environment to add their knowledge to it. This way, existing specialized subsystems and previously invisible or erroneous data are added. This results in a continuous process of ontology evaluation and expansion, in which eventually anyone may participate.

4.3 Interconnecting semantic models

How to approach the integration of different semantic models is a research topic of its own. Vdovjak et al. describe a RDF based architecture to provide this integration [28]. They distinguish between two general approaches to the matter: “In the data warehousing (eager) approach all necessary data is collected in a central repository before a user query is issued. This however, brings consistency and scalability problems. The on-demand driven (lazy) approach collects the data from the integrated sources dynamically during query evaluation”. Vdovjak et al. favor the latter approach.

We already saw the different nature of the components which will make up the semantic web applications in the near future. On one hand, we have a structured collection of facts with a focus on the concrete facts but not on how to interpret, relate or use them. When adding semantics to an otherwise syntactic and meaningless collection of structured data, we deviate from the detailed and application oriented environment of data modeling and work towards increasingly conceptual and abstract structures. However, this transformation is not straightforward. Simply transforming everything towards OWL, as we saw in the previous section, does not yet ensure that it will be a coherent logical knowledge base. The converted data base has a different nature than the converted domain model. And both are probably too specific to be called a true ontology, the most abstract of conceptual structures. We will first investigate this difference in conceptual model nature more closely in section 0, to gain a more complete understanding of the concerns to keep in mind when designing a future Semantic Web application. Finally, we will conclude this chapter in section 4.3.2, looking at how to connect a new application to the larger Semantic Web.

4.3.1 Conceptual model types

When working towards a global semantic data structure for use on the Semantic Web, we will analyze the nature of source and goal models. Dillon et al. provide an overview of the differences between modeling approaches [29]. As we saw before, the target of the Semantic Web is to reach a shared, agreed common conceptual structure. Because it is impossible to create such structure in one go for the whole world (if such structure is ever possible), the Web starts off with smaller domains and smaller levels of agreement about the representation and meaning of concepts and relations. Furthermore, almost all Semantic Web documents, like our own, have non-ontology predecessors. Since all is based upon extensible markup language, different views may be represented next to each other. After a while, some differences will be reconciled, resulting in a de facto standard view of the domain, while others may remain. However, it's good to keep in mind that the real target of ontology engineering is an abstract model that's commonly agreed upon by as many parties as possible. The most important properties of a real ontology identified by Dillon et al are as follows: both knowledge and its meaning should be agreed upon, the ontology should be shared and used, and it should be designed without a specific application in mind. Our welfare domain model is targeted to be such ontology. We chose to create a conceptual model, to restrain from implementation details which would not be recognized by other parties in the same domain. For example, another counseling organization will definitely use another supportive system with another underlying data structure. They will more easily agree upon concept and relation names than table and column names, or which columns are grouped. Since we are planning to deploy our semantic web model, it will be used. But the usage of a real ontology should be extended beyond the scope of one counseling organization. When the contents are satisfactory for a small isolated domain, new challenges await when seeking agreement with surrounding or comparable domains. Our model is not totally an ontology when we look at the general application requirement. Our Problem and Solution concepts are more targeted at the specific counseling application than more general concepts like Organization and Address. When we use the terminology of Dillon et al, our model looks like a mix between a knowledge base and an ontology. See Table 3 for the result of our analysis.

objectives	conceptual structure of welfare domain (ontological), but with some concepts targeted at particular states for counseling
consistency	most facts are always true; however, there are also facts that can only be verified or populated based on a particular state of affairs. The static part of our model can be regarded ontology, while the dynamic part looks more like a knowledge base.
actions	the model is targeted at problem solving. This aspect is therefore like a knowledge base.
knowledge	all stored knowledge is domain knowledge. Operational knowledge is not taken into account in the model, but is left outside as a population problem.

applicability	some concepts are applicable for a wider domain. Especially the static part, describing domain knowledge, is useable by a wide array of applications in the welfare environment. However, the concepts of Problem and Solution, that form the center of our model, are designed with our application background in mind.

Table 3 – Analysis of relationship of our model towards knowledge base and ontology, adapted from Dillon et al. **[29]**

This observation leads us to conclude that we cannot just transform our model to an ontology. We should rather identify the knowledge base parts in it and separate these from the ontology part. Dillon et al. call the knowledge stored in these two parts *operational* knowledge and *domain* knowledge, respectively. Consequently, creating an ontology for both types of knowledge would result in an operational ontology and a domain ontology. The domain ontology is a conceptual structure containing abstract and generally agreed concepts and relations. For example, concepts like Organizations, Products, Persons and Addresses may be reused based on their definition in a wider domain. New properties specific to the welfare environment may be added. But the domain ontology remains void of any reference to applications one may design around it. These references, as well as the link to specific conceptual (data) models used for a specific application in the welfare domain, will be put into the operational ontology. Different counseling organizations may want to use the same global information structure – they are after all operating in the same environment – but disagree upon the way this information should best be used. They may even disagree on what counseling is exactly. By designing their own operational ontology, as we have implicitly done in our model, each party may extend upon the shared part to create an interoperable semantic data structure.

Figure 23 shows a graphical overview of the described situation. As we see, a supporting (legacy) relational database is residing at the operational level, connected to its corresponding semantic model. This model might be originating from a domain model as the one we constructed in chapter 3, it might be a result of the transformation described in section 4.2.3, or perhaps most likely, a combination of both. The most generic concepts will become a shared welfare domain ontology, to which all operational ontologies may be connected. As the Semantic Web grows, the boundaries between different kinds of conceptual models will dissipate; everything will be part of a large, interconnected structure. However, the conceptual differences between parts of this model will remain. It will be worthwhile to keep in mind that in a mature Semantic Web all RDF triples are equal, but some are more equal than others.

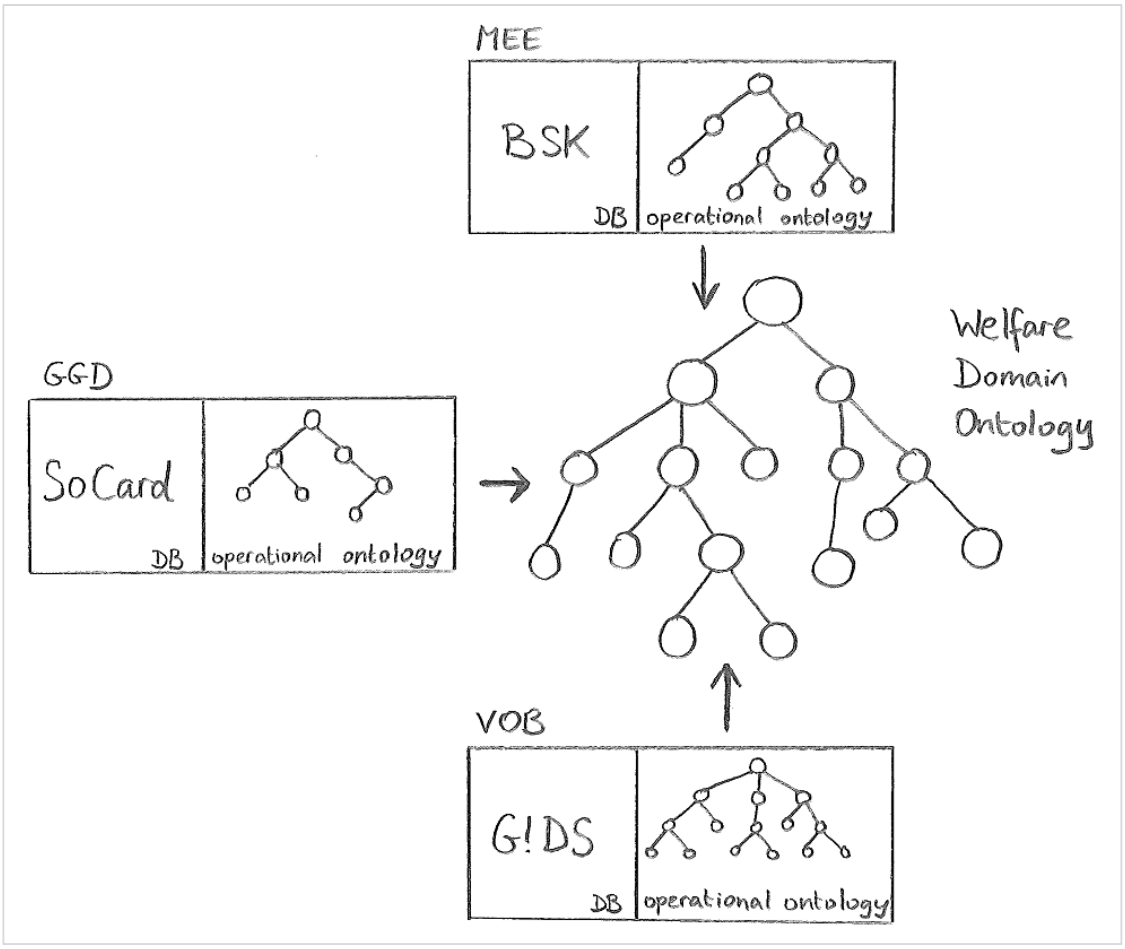

Figure 23 – Structuring of different conceptual models, adapted from Dillon et al. **[29]**

4.3.2 Connecting to the world

The semantic model will probably be related to other semantic models. Wherever possible, existing overlap can be utilized to minimize redundancy in the definition of concepts. This is important to avoid ambiguity and make it easy to understand other semantically annotated applications and also to be understood. However, should redundancy occur, it is not necessarily a problem. Constructing a mapping between two comparable definitions of a (high level) concept is usually feasible.
We assume that the constructed semantic concept structure will also be filled with instance data to actually create a data structure, which is interlinked with other sources of data. After all, we are mostly interested in actual data and not only in general concepts and their relations.

The Semantic Web will only become a global web if there are links between different smaller webs. Therefore, it is desirable to try to connect a new semantic application to existing structures where possible. When this connection is not present, navigating to related information, using derivations or combining different information sources becomes more difficult and less reliable. A first step may be to convert our own existing applications to a coherent web of semantic counterparts. But to facilitate a connection to the rest of the world, at least a global idea of existing structures is needed. To find these existing structures, we tried three presently available semantic search engines: SWSE[16], Swoogle[17] and Sindice[18]. The most promising results are shown in Table 4.

location	size
http://lat.inf.tu-dresden.de/~meng/ontologies/nciOntology.owl	32 MB
http://www.loa-cnr.it/ontologies/OWN/OWN.owl	24 MB
http://www.esd.org.uk/standards/lgcl/1.03/lgcl-schema	1,5 MB
http://www.smartweb-project.de/ontology/swinto0.3.1.rdfs	1 MB
http://dbpedia.org	> 103 mil

Table 4 – ontologies related to the welfare environment

While searching for available ontologies, we were disappointed by current results. There are some very large ontologies available, of which the DBpedia ontology is the best example. This ontology is generated from the very large user-maintained Wikipedia. It contains semantically structured fragments extracted from all kinds of Wikipedia pages. However, the relevance for our domain is very limited and doesn't come near the more than 103 million triples in the DBpedia ontology. Furthermore, a lot of semantically enriched web resources make use of common and very global ontology concepts, mainly from FOAF, Dublin Core and DBpedia. Some resources from the welfare domain are also expressible in concepts and properties found in these general ontologies. Based on this short investigation, we believe that currently a specific ontology may best be designed using the following steps:

- identify subsets from currently available widely used ontologies to use in the new ontology. This serves a dual purpose: we don't need to reinvent the wheel and we make sure our ontology is reusable for people familiar with these third party concepts.
- define our own concept structure ourselves, to keep it closely related to the way people think in our own domain and already reap the benefits of local semantic search.

We expect that the Semantic Web will grow like the normal Web did; once the Web is dense enough and every aspect of it has been polished and become common knowledge, using other semantic web pages and being used by other applications should be easier than it is now.

[16] http://swse.deri.org
[17] http://swoogle.umbc.edu
[18] http://sindice.com

5 Transforming PSM to PSL

As we concluded in section 4.2.1, current transformation methods from ORM towards (description) logics [17] [18] are not satisfactory in our opinion. This has two main reasons:

1. description logic languages are designed with a certain application area in mind, therefore they typically don't support expressing all constructs from a more generic language like ORM well. We argue that such transformation is overly cumbersome, since we don't want to choose from available logic languages at this level of abstraction. Rather, we propose to first transform the ORM model into generic logics and from there choose an appropriate more specific logic language when needed. Bridging the gap between generic logics and description logics is easier than bridging the gap between ORM and description logics in one go.
2. ORM transformation methods do not include PSM constructs, notably the power type and generalization. Furthermore, we noticed that the notion of objectification as covered by the analyzed transformation methods differs from our notion. The objectified n-ary fact type, split up into n binary fact types in the same way we will show in our method, is missing the external uniqueness constraint which is present in our version.

In this chapter, we introduce a transformation from ORM to a generic logics language, which we call PSL. This stands for Predicator Set Logics or Pretty Simple Logics, depending on what you prefer. Our point is that it is based on the Predicator Set Model and that it's pretty simple to understand, as was our main target when designing it. In our opinion, a transformation step towards the formal logics domain should be intuitive and quite straightforward, to minimize the probability errors or inconsistencies will be introduced. Preferably, the ORM model, when transformed to logics, should still be readable for a domain expert with some logics affinity.

The remainder of this chapter is structured as follows: first we introduce syntax elements specific to PSL in section 5.1. The transformation from PSM to PSL will then be performed in two steps. In section 5.2, we construct a transformation from the most important basic PSM constructs to PSL. In section 5.3, we show how to transform more advanced PSM constructs to their equivalent using only basic PSM constructs that can be handled by the method introduced in section 5.2. We conclude this chapter in section 5.4 with a small example illustrating our method.

5.1 PSL syntax and semantics

The PSL language makes use of default logical operators ($\rightarrow$, $\wedge$, $\vee$, $\neg$) and PSM expressions (as described in section 2.3.1), or more completely in [7]. The syntax and semantics of PSL are described in Table 5. In the running text, we will use "x is of type A" as the natural language semantics of the $x : A$ PSL construct.

syntax	Semantics
$x : A$	$x \in \mathsf{Pop}\ (A)$
$x[\tau]$	return element τ contained in x
$x\{\tau\}$	select subset τ contained in x
$f \triangleq D$	f is defined as D
$\rightarrow$	logical implication (rule level)
$\wedge$	logical and (rule level)
$\vee$	logical or (rule level)
$\neg$	logical not (rule level)

Table 5 – PSL syntax and semantics

5.2 *Transforming basic PSM to PSL*

We define the following PSM constructs as the basic PSM constructs:

- B1: fact type
- B2: specialization
- B3: generalization
- C1: uniqueness constraint
- C2: mandatory constraint
- C3: set constraints

The B-type constructs are basic PSM building blocks, while the C-type constructs are basic PSM constraints. Note that this categorization does not indicate the level of schema complexity which may be reached using only these constructs. The following definition explains the difference between advanced and basic PSM constructs: *an advanced PSM construct is a PSM construct that can be transformed in a different PSM construct with the same meaning using only basic PSM constructs*. We will treat these constructs in section 5.3.

5.2.1 Fact type

When considering a fact type x, we are interested in the interpretation of $x : F$. A fact type in PSM is a set of roles which draw their population from an associated base entity type. It may be retrieved by using the PSM Base function. Consequently, we may define the PSL rule associated to fact types as follows:

[PSL1] $x : F \rightarrow \forall_{r \in x} \left[x[r] : \mathsf{Base}(r) \right]$

5.2.2 Specialization

When entity type *A* is a specialization (or subtype) of entity type *B*, this is expressed in PSM as *A* Spec *B*. The population of *A* is not assigned to it directly, but rather derived from the population of *B*. The subtype defining rule *P* acts as a filtering condition upon the population of *B*. This leads to the following PSL rule:

[PSL2] $x : B \wedge P(x) \leftrightarrow x : A$

The specialization rule is twofold: firstly, we know that any instance *x* which is of type *A* is an instance of type *B* for which subtype defining property *P* holds. Secondly, this rule also holds the other way around since every instance *x* of type *B* for which property *P* holds is also of type *A*.

5.2.3 Generalization

A generalization occurs when one or more entity types are put together into a newly formed entity type. When *n* entity types generalize to entity type *B*, this may be expressed as A_1 Gen *B*, A_2 Gen *B*, ..., A_n Gen *B*. For an instance *x* having type *B*, with *B* a generalization of one or more other entity types, we have this PSL rule:

[PSL3] $x : B \rightarrow \exists_A \left[A \text{ Gen } B \wedge x : A \right]$

5.2.4 Uniqueness constraint

A basic uniqueness constraint spans $i \leq n$ roles of a *n*-ary fact type. Typically, *i* is equal to *n* – 1 or *n*. This fact type may simply be an element of F, as we have seen in section 5.2.1, but it may also be composed of different fact types in case of an external uniqueness constraint. We will see an example of such uniqueness constraint in section 5.3. For now, we assume there is a Uniquest function present that has the following definition:

$\text{Uniquest}(\tau) = f$

with: $\tau \triangleq$ the set of roles forming the uniqueness constraint;

$f \triangleq$ the result of the join via common object types of the fact type set Φ

$\Phi \triangleq \left\{ \varphi \in F \;\middle|\; \exists_{r \in \tau} \left[\text{Fact}(r) = \varphi \right] \right\}$

The Uniquest algorithm, outlined in the overview article by Van der Weide et al. [30], implements this function. We refer to this article for further details. Note that the Uniquest function is always one fact type *f*, of which τ is a subset. Using these *f* and τ, we can define a PSL rule capturing the uniqueness constraint.

[PSL4] $\text{Unique}(\tau) \leftrightarrow \forall_{x,y} \left[(x : \text{Uniquest}(\tau) \wedge y : \text{Uniquest}(\tau) \wedge x\{\tau\} = y\{\tau\}) \rightarrow x = y \right]$

We introduce the shorthand notation Unique(τ) to express that τ is a uniqueness constraint according to PSL4. This rule states that two instances *x* and *y* of the fact type which is constrained by UC τ are in fact the same instance when they don't differ from each other regarding only the roles captured by the UC. This is exactly what uniqueness means in PSM. Note that by evaluating the Uniquest algorithm for τ, we achieve that every valid uniqueness constraint is covered in one go.

5.2.5 Mandatory constraint

We now look at the mandatory constraint, using the shorthand notation Mandatory(τ). This constraint may be defined for one role as well as multiple roles, just like the uniqueness constraint. However, all these roles must share a *common base* for the mandatory constraint to be valid. This requirement for τ is expressed as follows:

τ is a valid argument in Mandatory(τ) iff $\forall_{r_1,r_2\in\tau}\left[\text{Base}(r_1)=\text{Base}(r_2)\right]$.

CommonBase(τ) returns the common base of all roles in such τ.

Given a mandatory constraint with a valid τ like described above, we may also formulate the corresponding PSL rule as follows:

[PSL5] $\text{Mandatory}(\tau)\leftrightarrow\forall_x\left[x:\text{CommonBase}(\tau)\rightarrow\exists_{r\in\tau}\exists_f\left[f:\text{Fact}(r)\wedge f[r]=x\right]\right]$

To put PSL5 in natural language: for all instances *x* from the population of the common base, there should be a corresponding fact type instance containing a role from the mandatory constraint in which x occurs as an instance. This is the meaning of a mandatory constraint as intended in PSM.

5.2.6 Set constraints

The last constraint type we will cover is the set constraint, which is between two sets of roles with equal population capabilities. The constraint *C*(σ,τ) includes a mapping function Φ between both sets of roles σ and τ. We assume this mapping to be present in all cases. Φ may be given a role from σ or τ as an argument and it will return the associated mapped role from τ or σ, respectively. There are three different kinds of set constraint: subset, equality and exclusion. We will treat them subsequently.

5.2.6.1 Subset constraint

The notation Subset(σ, τ) means that the population of roles in σ should always be a subset of the population of corresponding roles in τ. Expressed in PSL, this looks like:

[PSL6] $\text{Subset}(\sigma,\tau)\leftrightarrow\forall_{r\in\sigma}\forall_x\left[x:r\rightarrow\exists_{s\in\tau}[\Phi(r)=s\wedge x:s]\right]$

5.2.6.2 *Equality constraint*

The notation Equal(σ, τ) means that the population of roles in σ should always be equal to the population of corresponding roles in τ. This is equivalent to a bidirectional subset constraint. Therefore, it may be expressed as follows:

[PSL7] $\text{Equal}(\sigma,\tau) \leftrightarrow \text{Subset}(\sigma,\tau) \wedge \text{Subset}(\tau,\sigma)$

5.2.6.3 *Exclusion constraint*

The notation Exclusion(σ, τ) means that the population of roles in σ should never occur in the population of corresponding roles in τ. The logical interpretation is expressed in PSL8 below.

[PSL8] $\text{Exclusion}(\sigma,\tau) \leftrightarrow \forall_{r\in\sigma}\forall_{x}\left[x:r \rightarrow \exists_{s\in\tau}[\Phi(r)=s \wedge \neg(x:s)]\right]$

5.3 Transforming advanced PSM to basic PSM

With the constructs in section 5.2, a simple PSM schema can already be interpreted logically. However, we are missing some essential constructs which we did in fact use in our own welfare domain model. We will treat the three most important remaining constructs in this section:

- T1: objectification
- T2: power type
- T3: sequence type

These constructs will all be transformed to a semantically equivalent PSM model which only uses basic PSM constructs. Consequently, any general PSM schema Σ may be transformed to PSL by first applying the rules in this section, leading to a projection Σ', and then applying rules from section 5.2 to Σ'.

5.3.1 Objectification

An objectified fact type is of course still a fact type, but besides this it also plays the role of an object type in another fact type. Therefore, it may be defined as follows:

$\text{Objectified}(f) \triangleq f \in F \wedge \exists_{x}\left[\text{Base}(x)=f\right]$

For all fact types *f* for which Objectified(*f*) holds, we are going to introduce a derived object type taking its place in relation to other fact types. In fact, this transformation is analogous to the one performed by Jarrar [16] and Keet [17]. However, compared to their transformation we add an extra external uniqueness constraint to capture the full meaning of the fact type in Σ in the projection Σ' as well.

Let $f = \{r_1, r_2, \ldots, r_n\}$ be a *n*-ary fact type in Σ $(n \geq 2)$ for which Objectified(*f*) holds.

We introduce binary fact types $\varphi(r_1), \varphi(r_2), \ldots, \varphi(r_n)$ with $\varphi(r_i) = \{r_{i1}, r_{i2}\}$ and an entity type f_O, such that the following properties hold for every fact type $\varphi(r_i)$ (with $1 \leq i \leq n$):

(1) $\mathsf{Base}(r_{i1}) = f_O$

(2) $\mathsf{Base}(r_{i2}) = \mathsf{Base}(r_i)$

(3) $\mathsf{Mandatory}(\{r_{i1}\})$

(4) $\mathsf{Unique}(\{r_{i1}\})$

(5) $\mathsf{Unique}(\{r_{12}, r_{22}, \ldots, r_{n2}\})$

Additionally, the Base function is modified as follows:

(6) For every role *x*, if $\mathsf{Base}(x) = f$ in Σ, $\mathsf{Base}(x) = f_O$ in Σ'.

In this method, we create an additional object type f_O of which the instances may be directly addressed. Each instance corresponds to an instance of the fact type *f* it reflects. The reflection is represented using *n* binary fact types, relating all roles within *f* to the new object type. The mandatory and uniqueness constraints ensure that this projection has the same behavior as the original fact type (i.e. all roles are being part of of it exactly one time and any two instances should be different on at least one role).

5.3.2 Power type

A power type *A* containing an element type *B* can be represented as a simple binary relationship with an additional specific power type constraint. The transformation is as follows:

Let *A* be a power type containing element type *B.*

We introduce a fact type $f_E = \{r_P, r_E\}$ such that:

(1) $\mathsf{Base}(r_P) = A$, $\mathsf{Base}(r_E) = B$

(2) $\mathsf{Mandatory}(r_P)$

The fact type f_E may be populated according to the population of *A* and *B*. Every instance $a \in A$ contains *n* elements from *B* and has the form $\{b_1, b_2, \ldots, b_n\}$ $(n \geq 1)$. This leads to *n* instances of f_E of the form $\{a, b_i\}$ $(1 \leq i \leq n)$. As two instances a_1 and a_2 cannot both consist of exactly the same elements from *B* in Σ, this requirement should also be reflected in Σ'. We do this by adding the following rule:

[PSL9] $\forall_b \left[\{a_1, b\} : f_E \leftrightarrow \{a_2, b\} : f_E \right] \rightarrow a_1 = a_2$

Rule PSL9 is a rather abstract logical rule, which ensures that when two elements a_1 and a_2 are equal when they behave the same with respect to all instances of *B*. The left hand side of the implication only evaluates to true when all $b \in B$ are either related to both a_1 and a_2 or not related to both a_1 and a_2. If that is the case, we have a reflection of an a_1 and a_2 that have exactly the same elements from *B*. This is only legal when a_1 and a_2 are in fact the same element.

5.3.3 Sequence type

The sequence type is converted the same way as a power type, but with an extra indexing fact type related to an index label type *I* and the objectified element fact type f_{E_O}. Using the rules from section 5.3.2 and 5.3.1, we can already handle this transformation.

5.4 *Transformation example*

In this section, we will illustrate our transformation method with a concrete example, taken from the welfare domain. For this example, we use the PSM model about describing texts (found in Figure 15) as a source, since it contains many special constructs and because it is a quite isolated part of the larger model. In section 5.4.1, we will first define a sample population associated with the general model for use in this example. From this model, we will choose a typical example instance. Thereafter, we will transform the necessary advanced constructs to basic PSM in section 5.4.2. Finally, we will transform some exemplary parts to PSL in section 5.4.3.

5.4.1 Sample population

The sample population is based on the Living example found in Figure 14. For clarity, we reformulate it to its PSM counterpart. Figure 14 breaks up as follows: there are two kinds of basic objects: DTextNodes and DTextLeafs. The leafs are marked with a filled square, marking their associated DText. They are populated with *(x,y)* tuples according to the PSM schema:

- For a DTextLeaf according to "Title *x* labeling DText *y*".
- For a DTextNode according to "Title *x* labeling DTextTree *y*".

All DTextLeaf and DTextNode instances are generalized to be DTextSubtrees. Therefore, the DTextSubtree population contains the following 11 elements:

st1:	('Question 1?', dt1)	st7:	('Living', dtt1)
st2:	('More info', dt2)	st8:	('At home', dtt2)
st3:	('Products', dt3)	st9:	('Help', dtt3)
st4:	('People', dt4)	st10:	('Somewhere else', dtt4)
st5:	('Question 2?', dt5)	st11:	('Waiting time', dtt5).
st6:	('Speeding up', dt6)		

Note that 'st' stands for DTextSubtree, 'dt' for DText and 'dtt' for DTextTree.
As we see, the subtree pool is composed of six DTextLeafs and five DTextNodes, all of which may be placed in any larger DTextTree. DTexts which are required to have an associated DTextLeaf, may also have related ThesaurusTerms; an example population for this is simply one tuple, coming from "DText dt1 described by ThesaurusTerm 'home'". The DTextTree elements are sequences of subtrees, specified as follows:

dtt1:	[st8, st10]
dtt2:	[st1, st2, st9]
dtt3:	[st3, st4]
dtt4:	[st11]
dtt5:	[st5, st6]

Note that the only DTextSubtree that is not included in a DTextTree is st7, which happens to be the root subtree. By rule ORC6, we can indeed derive that st7 is a Root.

5.4.2 Transformation to basic PSM

We will not transform the total population to PSL, but rather take one intesting part of it: dtt2 and its properties. This is a *sequence type*, so we should apply the rule found in section 5.3.3. We construct the membership fact type $f_E = \{r_P, r_E\}$ and fill its population accordingly. This leads to three tuples: (dtt2, st1), (dtt2, st2) and (dtt2, st9). An index needs to be assigned to all of these tuples. For this, we objectify F_E to create the object type F_{E_O} and the fact types $\varphi(r_P)$ and $\varphi(r_E)$, using the method in section 5.3.1. The converted population is as follows (including the fact type F_I relating index and F_{E_O} to each other).

F_{E_O}	$\varphi(r_P)$	$\varphi(r_E)$	F_I
t1	(t1, dtt2)	(t1, st1)	(t1, 0)
t2	(t2, dtt2)	(t2, st2)	(t2, 1)
t3	(t3, dtt2)	(t3, st9)	(t3, 2)

So instead of dtt2 being a sequence type, we now have three simple binary fact types and a set of constraints describing dtt2. This is all still in ORM. We omit the details concerning the constraints here; they are in the boxes of section 5.3.1 and 5.3.2. The fact types constituting the sequence type may be simple, but the same cannot be said about the object types. First of all, we have the DTextSubtree object, which is a generalization of the *fact types* DTextLeaf and DTextNode. They are both objectified in Figure 15; according to our PSL definition a fact type *f* is objectified when there is a role *x* with Base(*x*) = *f*. This doesn't *seem* the case in Figure 15, but in the breakup of the sequence type we see that DTextNode and DTextLeaf are both base of a role in $\varphi(r_E)$. After generalization, that is. We therefore have to derive DTextNode_O and DTextLeaf_O in the same way like we did it inside the sequence type. We don't go through all steps here again, but we present the result at once:

DTextLeaf_O	$\varphi_L(r_1)$	$\varphi_L(r_2)$
st1	(st1, 'Question 1?')	(st1, dt1)
st2	(st2, 'More info')	(st2, dt2)

DTextNode_O	$\varphi_N(r_1)$	$\varphi_N(r_2)$
st9	(st9, 'Help')	(st9, dtt3)

Note that we already labeled the tuples according to the label we chose for the generalized DTextSubtree instances for readability, although technically this is not needed. All instances featured in the objectified fact types DTextNode_O and DTextLeaf_O together form the population of the DTextSubtree object type.

5.4.3 Transformation to PSL

5.4.3.1 Fact types

All fact types we saw may be easily transformed to PSL in the same manner. We will transform one fact type to get the idea:

general rule for $\varphi(r_E)$: $x:\varphi(r_E)\to\forall_{r\in x}\left[x[r]:\text{Base}(r)\right]$

$(t_1,st_1):\varphi(r_E)\to\forall_{r\in(t_1,st_1)}\left[(t_1,st_1)[r]:\text{Base}(r)\right]$ leading to 1) $t_1:F_{E_O}$ 2) $st_1:\text{DTextSubtree}$

$(t_2,st_2):\varphi(r_E)\to\forall_{r\in(t_2,st_2)}\left[(t_2,st_2)[r]:\text{Base}(r)\right]$ leading to 1) $t_2:F_{E_O}$ 2) $st_2:\text{DTextSubtree}$

$(t_3,st_9):\varphi(r_E)\to\forall_{r\in(t_3,st_9)}\left[(t_3,st_9)[r]:\text{Base}(r)\right]$ leading to 1) $t_3:F_{E_O}$ 2) $st_9:\text{DTextSubtree}$

In the same manner, we may derive $st_1:\text{DTextLeaf}_O$ and $st_9:\text{DTextNode}_O$ from the transformation of the DTextLeaf and DTextNode fact types.

5.4.3.2 Generalization

The dual typing of st_1 and st_9 is due to the generalization, as we can also verify using PSL3. Clearly, this rule is valid for all our DTextSubtree instances.

$x:\text{DTextSubtree}\to\exists_A\left[A\ \text{Gen DTextSubtree}\wedge x:A\right]$

DTextLeaf_O Gen DTextSubtree and DTextNode_O Gen DTextSubtree

5.4.3.3 Specialization

As an example, we will treat two of the specializations in Figure 15: Root and Subject. Recall the subtype defining rules ORC6 and ORC7:

[ORC6]	LET Root BE (DTextNode BUT NOT OCCURRING-IN DTextTree)
[ORC7]	LET Subject BE (Title of DText)

ORC rules need to be manually transformed to PSL now. However, an automatic mapping is reachable with future research. In PSL, these rules are as follows:

$$\text{Root}(x) \triangleq x : \text{DTextNode} \wedge \neg\exists_y[(y,x) : \varphi(r_E)]$$

$$\text{Subject}(x) \triangleq x : \text{Title} \wedge \exists_y[(x,y) : \text{DTextLeaf}]$$

When we would evaluate these rules using our example population, we would get the following truth tables, corresponding to PSL2, as a result. Note that we only consider instances matching the first condition and an exemplary instance that does not.

	x : B	*P(x)*	*x : A*
x	x : DTextNode	$\neg\exists_y[(y,x) : \varphi(r_E)]$	Root(x)
st6	0	0	**0**
st7	1	1	**1**
st8	1	0	**0**
st9	1	0	**0**
st10	1	0	**0**
st11	1	0	**0**

	x : B	*P(x)*	*x : A*
x	x : Title	$\exists_y[(x,y) : \text{DTextLeaf}]$	Subject(x)
dt1	0	1	**0**
‘Question 1?’	1	1	**1**
‘More info’	1	1	**1**
‘Products’	1	1	**1**
‘People’	1	1	**1**
‘Question 2?’	1	1	**1**
‘Speeding up’	1	1	**1**
‘Living’	1	0	**0**
‘At home’	1	0	**0**
‘Help’	1	0	**0**
‘Somewhere else’	1	0	**0**
‘Waiting time’	1	0	**0**

Note that the subtype specialization rules define both parts of the PSL2 right hand side and that the specialization type defines the left hand side.

5.4.3.4 Constraints

We finish up this example section with an example for all three constraints found in Figure 15: the mandatory constraint and exclusion constraint on both "Title labeling..." roles, and the external uniqueness constraint within the sequence type. These constraints can be expressed in PSL according to our transformation rules, as follows:

$$\text{Unique}(\tau) \leftrightarrow \forall_{x,y}\left[(x:\text{Uniquest}(\tau) \wedge y:\text{Uniquest}(\tau) \wedge x\{\tau\} = y\{\tau\}) \rightarrow x = y\right]$$

In our example, $\tau = \{r_{P_2}, r_{E_2}\}$. The result of Uniquest(τ) is a quadruple fact type in this case, containing roles associated to F_{E_O}, DTextTree, DTextSubtree and Index. When we restrict ourselves to the example sequence type dtt2, we get the following population:

r	r_1	r_{P_2}	r_{E_2}	r_I
Base(r)	F_{E_O}	DTextTree	DTextSubtree	Index
x_1	t1	dtt2	st1	0
x_2	t2	dtt2	st2	1
x_3	t3	dtt2	st9	2

In the population table, τ is colored gray. This uniqueness constraint states that if we take any two rows from the table, they should be different when only taking the gray columns into account. If two rows are the same when only considering the gray columns, they should be the same as a whole. As our population adheres to this rule, there are no double rows offending it.

We transformed both objectified fact types DTextLeaf and DTextNode to their basic PSM counterparts. Now, since we are going to look at the mandatory and exclusion constraints on the roles which have Title as their Base, we are only interested in sub fact types $\varphi_L(r_1)$ for the DTextLeaf part and $\varphi_N(r_1)$ for the DTextNode part. To be able to separate them, we define $\varphi_L(r_1) = \{r_{L_1}, r_{L_2}\}$ and $\varphi_N(r_1) = \{r_{N_1}, r_{N_2}\}$. Now, we can define τ for the mandatory constraint: $\tau = \{r_{L_1}, r_{N_1}\}$, with CommonBase(τ) = Title.

$$\text{Mandatory}(\tau) \leftrightarrow \forall_x\left[x:\text{CommonBase}(\tau) \rightarrow \exists_{r\in\tau}\exists_f\left[f:\text{Fact}(r) \wedge f[r] = x\right]\right]$$

The mandatory constraint is only valid if for all *x* in Title we can find a role in τ that has an associated fact type instance containing *x*. Let's look at the table summing up these roles, validating the constraint.

x : Title	$r \in \tau$	Fact(r)	$f\,[\,f$: Fact(r) $\wedge f\,[r] = x\,]$
'Question 1?'	r_{L_1}	$\varphi_L(r_1)$	(st1, 'Question 1?')
'More info'	r_{L_1}	$\varphi_L(r_1)$	(st2, 'More info')
'Products'	r_{L_1}	$\varphi_L(r_1)$	(st3, 'Products')
'People'	r_{L_1}	$\varphi_L(r_1)$	(st4, 'People')
'Question 2?'	r_{L_1}	$\varphi_L(r_1)$	(st5, 'Question 2?')
'Speeding up'	r_{L_1}	$\varphi_L(r_1)$	(st6, 'Speeding up')
'Living'	r_{N_1}	$\varphi_N(r_1)$	(st7, 'Living')
'At home'	r_{N_1}	$\varphi_N(r_1)$	(st8, 'At home')
'Help'	r_{N_1}	$\varphi_N(r_1)$	(st9, 'Help')
'Somewhere else'	r_{N_1}	$\varphi_N(r_1)$	(st10, 'Somewhere else')
'Waiting time'	r_{N_1}	$\varphi_N(r_1)$	(st11, 'Waiting time')

The exclusion constraint is also concerning both roles r_{L_1} and r_{N_1}, but this time they should be put in a separate set of one instance: $\sigma = \{r_{L_1}\}$ and $\tau = \{r_{N_1}\}$. The mapping function Φ is clear: only one mapping is possible in this case. $\Phi(r_{L_1}) = r_{N_1}$, and that's the only case we need to evaluate. Since there is only one element in σ and τ, we may simplify the Exclusion constraint PSL rule to:

$$\text{Exclusion}(\{r_{L_1}\},\{r_{N_1}\}) \leftrightarrow \forall_x \left[\, x : r_{L_1} \rightarrow \neg\,(x : r_{N_1})\,\right]$$

As we can see in the table above, all Title instances are assigned to either r_{L_1} or r_{N_1}. This constraint enforces that no x may populate both roles at the same time. Note that the exclusion constraint is bidirectional, so we may also reverse r_{L_1} and r_{N_1}.

This example shows the transformation of only a tiny bit of the total population of our PSM schema. It should be possible to perform it automatically once appropriate tools are available. This is a realistic scenario, given that there already exists a plugin for the DogmaModeler tool that (largely) implements the Jarrar transformation method from ORM to Description Logics [18].

6 Conclusion and future work

6.1 Conclusion

After analyzing the demands of a Semantic Web application and the current application development approach in the welfare domain, we found there is an apparent abstraction gap between the activities in the real environment and the functionality of current applications. Because there often is no formal bridge between the real world and supporting applications, deploying such applications in the Semantic Web is difficult. This led to our main question: *"How should the abstraction gap between environment and application be bridged?"*

The desired bridge is essentially a sound and complete formalization process, translating knowledge present in the real welfare domain – or any other domain – to a formal model which is suitable for formal reasoners. Such model may be deployed in the Semantic Web without much effort.

The result of our approach is a PSM/ORC formalization using structured natural language, ensuring that people from the domain are able to relate their real world activities to corresponding activities in the model. At the same time, this formalization ensures that the model is suitable to be interpreted by logics reasoners, as we have shown in chapter 5 by transforming it to a generic logics language called PSL. Together, the PSM/ORC model and its PSL interpretation provide Semantic Web application developers with a knowledge model which is formal enough to serve as a generic base, while still allowing different application choices to be made. In fact, these choices will have to be made, since there is an expressivity versus computability tradeoff to be handled. Since this is an application specific concern, we did not further elaborate it.

The Semantic Web is evolving, with the establishment of standards in domain ontologies on one hand and web application languages on the other hand. However, we believe that any successful knowledge-intensive application, such as the Social Map we investigated in our case study, needs a generic underlying semantic framework as we showed in our approach. Only when there is a thorough formal conceptual mapping between the environment and its applications, the vision of the Semantic Web will become reality.

6.2 Future work

Concerning the mapping between conceptual languages like PSM and ORM on one hand and actual logics reasoners on the other hand, some work has already been done, notably by Jarrar [16] [18] and Keet [17]. The main conclusion from their work is that a complete mapping to Description Logics is not possible. However, it is not yet clear how to determine which DL language should be chosen when designing an actual application based on a conceptual model. We chose to abandon this decision for now and construct a general mapping to logics. For an actual application, the tradeoff between needed expressivity and computability should be investigated. This might be done at a general level or ad hoc.

We did not further address the challenge of populating Deliverable Solutions (see section 3.2.2.3). This challenge is an interesting future research topic, since dynamic knowledge dependencies will be occurring quite often in the future Semantic Web. This challenge is now mostly addressed by human effort, but using a machine-interpretable knowledge base and applicable techniques (e.g. from artificial intelligence) applications may become able to assist in this process.

References

[1] T. Berners-Lee, R. Cailliau, A. Luotonen, H. Frystyk Nielsen, and A. Secret, "The World Wide Web," *Communications of the ACM*, vol. 37, no. 8, pp. 76-82, 1994.

[2] T. Berners-Lee, "Semantic Web Road map," Sep. 1998.

[3] A. Ankolekar, M. Krötzsch, T. Tran, and D. Vrandedic, "The Two Cultures," Institut AIFB, University of Karlsruhe (TH), 2007.

[4] J. Cardoso, "The Semantic Web Vision: Where are We?," *IEEE Intelligent Systems*, pp. 22-26, Sep. 2007.

[5] T. P. v. d. Weide, "Modeling and Reasoning," Radboud University Lecture Notes, version 01-12-2007.

[6] T. Halpin, "A logical analysis of information systems: static aspects of the data-oriented perspective," PhD Thesis, University of Queensland, 1989.

[7] A. t. Hofstede, H. Proper, and T. v. d. Weide, "Formal definition of a conceptual language for the description and manipulation of information models," *Information Systems*, vol. 18, pp. 489-523, 1993.

[8] L. A. Zadeh, "Knowledge Representation in Fuzzy Logic," *IEEE Transactions on Knowledge and Data Engineering*, vol. 1, no. 1, pp. 89-100, 1989.

[9] J. v. Ossenbruggen, L. Hardman, and L. Rutledge, "Hypermedia and the Semantic Web: A Research Agenda," *Journal of Digital Information*, vol. 3, no. 1, pp. 1-18, May 2002.

[10] I. Rahwan, "Mass Argumentation and the Semantic Web," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 6, no. 1, pp. 29-37, 2008.

[11] L. Ding, L. Zhou, T. Finin, and A. Joshi, "How the Semantic Web is Being Used: An Analysis of FOAF Documents," in *Proceedings of the 38th Annual Hawaii International Conference*, 2005.

[12] J. Ward, "A Quantitative Analysis of Unqualified Dublin Core Metadata Element Set Usage within Data Providers Registered with the Open Archives Initiative," in *Proceedings of the Joint Conference on Digital Libraries*, 2003, pp. 315-317.

[13] G. Antoniou and F. v. Harmelen, *A Semantic Primer*, 2nd ed. MIT Press, 2008.

[14] A. Harth and S. Decker, "Optimized Index Structures for Querying RDF from the Web," in *Third Latin American Web Congress*, 2005, pp. 1-10.

[15] L. Ding, et al., "Swoogle: A Search and Metadata Engine for the Semantic Web," in *Proceedings of the 13th ACM international conference on Information and knowledge management*, New York (NY), USA, 2004, pp. 652-659.

[16] M. Jarrar, "Towards Automated Reasoning on ORM Schemes," in *Proceedings of the 26th International Conference on Conceptual Modeling*, Heidelberg, Germany, 2007, pp. 181-197.

[17] C. M. Keet, "Mapping the Object-Role Modeling language ORM2 into Description Logic language DLRifd," *Arxiv preprint cs/0702089*, 2007.

[18] M. Jarrar, "Mapping ORM into the SHOIN/OWL Description Logic," *Lecture Notes in Computer Science*, vol. 4805, 2007.

[19] M. v. Assem, V. Malaisé, A. Miles, and G. Schreiber, "A Method to Convert Thesauri to SKOS," in *ESWC*, 2006, pp. 95-109.

[20] B. Motik, I. Horrocks, and U. Sattler, "Bridging the Gap Between OWL and Relational Databases," in *Proceedings of the 16th international conference on World Wide Web*, New York, NY, USA, 2007, pp. 807-816.

[21] C. Bizer and A. Seaborne, "D2RQ - Treating Non-RDF Databases as Virtual RDF Graphs," in *Proceedings of the 3rd International Semantic Web Conference*, 2004.

[22] J. Carroll, et al., "Jena: implementing the semantic web recommendations," in *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, New York, NY, USA, 2004, pp. 74-83.

[23] M. Svihla and I. Jelinek, "Benchmarking RDF Production Tools," *Lecture Notes in Computer Science*, vol. 4653, p. 700, 2007.

[24] J. Broekstra, A. Kampman, and F. v. Harmelen, "Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema," *Lecture Notes in Computer Science*, vol. 2342, pp. 54-68, 2002.

[25] L. Stojanovic, N. Stojanovic, and R. Volz, "Migrating data-intensive web sites into the Semantic Web," in *Proceedings of the 2002 ACM symposium on Applied computing*, New York, NY, USA, 2002, pp. 1100-1107.

[26] C. Hu, H. Li, X. Zhang, and C. Zhao, "Research and Implementation of Domain-Specific Ontology Building from Relational Database," in *The Third ChinaGrid Annual Conference*, Dunhuang, Gansu, China, 2008, pp. 289-293.

[27] N. Cullot, R. Ghawi, and K. Yétongnon, "DB2OWL: A Tool for Automatic Database-to-Ontology Mapping," in *Proceedings of the 15th Italian Symposium on Advanced Database Systems*, Torre Canne, Italy, 2007, pp. 491-494.

[28] R. Vdovjak and G.-J. Houben, "RDF Based Architecture for Semantic Integration of Heterogeneous Information Sources," in *Workshop on Information Integration on the Web*, 2001, pp. 51-57.

[29] T. Dillon, E. Chang, M. Hadzic, and P. Wongthongtham, "Differentiating Conceptual Modelling from Data Modelling,Knowledge Modelling and Ontology Modelling and a Notation for Ontology Modelling," in *Proceedings of the fifth on Asia-Pacific conference on conceptual modelling-Volume 79*, Wollongong, Australia, 2008, pp. 7-17.

[30] T. P. v. d. Weide, A. H. M. t. Hofstede, and P. v. Bommel, "Uniquest: Determining the Semantics of Complex Uniqueness Constraints," *The Computer Journal*, vol. 35, no. 2, pp. 148-156, 1992.