

**Ragnhild Holgersen**

---

**Prototyping a Decision Support System  
Based on Semantic Web Technologies to  
Aid Consumers with Food Sensitivities  
in their Assessment of Product Safety**

**Master thesis 2013  
Master in Library and Information Science  
Oslo and Akershus University College of Applied Sciences,  
Department of Archivalistics, Library and Information Sciences**

## Abstract

The thesis depicts the information need people with food sensitivities experience in the shopping situation. Diversity within the group is illustrated through five personas. Existing measures aimed at helping affected individuals in their search for safe food – ranging from labeling requirements and practices to various forms of information systems – are exemplified. Shortcomings of existing solutions are discussed from an information science perspective. An alternative approach based on Semantic Web technologies and Linked data is proposed, underpinning a decision support system. The Boolean interpretation of food safety is rejected in favor of a division that accounts for the need for human assessment of uncertain cases. SPARQL and automatic inference, relying on a dataset holding facts about allergen occurrence in various ingredients, is used to automatically classify products as safe, uncertain or unsafe for individual users. Proof of concept for the proposed approach is provided through a prototype web application. The automated classification is used to communicate the safety of each product, using familiar traffic light colors. Tailored decision support is provided on-demand, emphasizing information that is likely to impact the users assessment of uncertain products while limiting “information overload”. The development process behind the ontology and web application is discussed in detail, followed by a discussion about how to establish the required data sets.

## Sammendrag

Masteroppgaven skildrer det informasjonsbehovet mennesker med matoverfølsomhet opplever når de skal handle mat. Mangfoldet innen gruppen illustreres gjennom fem ”personas”. Eksisterende tiltak med hensikt å hjelpe personer med matoverfølsomhet i jakten på trygg mat – fra merkeregler og praksiser til forskjellige former for informasjonssystemer – eksemplifiseres. Svakheter ved eksisterende løsninger drøftes ut fra et informasjonsvitenskapelig perspektiv. En alternativ tilnærming basert på Semantisk Web teknologier og Lenkede Data introduseres og danner grunnlaget for et beslutningsstøttesystem for personer med matoverfølsomhet. Den Boolske tilnærmingen til mattrygghet forkastes til fordel for en tredeling som ivaretar behovet for skjønnsmessig vurdering i usikre tilfeller. SPARQL og automatisk inferens, basert på en kjerne av data om allergenforekomst i ingredienser, benyttes til å klassifisere produkter som trygge, usikre og utrygge. ”Proof of concept” for den foreslåtte tilnærmingen oppnås ved å prototype en webapplikasjon. Den automatiske klassifikasjonen blir brukt til å kommunisere i hvilken grad produkter er antatt å være trygge for den enkelte, ved bruk av kjente trafikklysfarger. Skreddersydd beslutningsstøtte tilbys, der informasjon som trolig vil påvirke brukerens manuelle vurdering av usikre produkter fremheves, mens andre opplysninger holdes tilbake for å unngå støy. Utviklingsprosessen bak ontologi og webapplikasjon drøftes inngående, etterfulgt av en diskusjon av hva som må til for å skaffe til veie og kvalitetssikre de datasettene som modellen baserer seg på.

**Oslo and Akershus University College of Applied Sciences,  
Department of Archival, Library and Information Sciences**

**Oslo 2013**

## PREFACE

My perspective on library and information science is influenced by the fact that I have a multidisciplinary educational background, incorporating courses from computer science and entrepreneurship. I believe that insights from the library and information sciences have applications far beyond the traditional library setting. People continually experience information needs in all kinds of everyday situations and increasingly rely on information technology as an aid. Thus, the need for efficient information systems, based on a thorough understanding of users' information needs and behaviors, is evident.

Growing up in a family where both milk allergy and severe nut allergy were represented, I myself was diagnosed with coeliac disease as an adult. This background has made me painfully aware of the problems that arise in the grocery-shopping situation as a consequence of food sensitivities.

Although food labeling is highly regulated and industry actors take various measures to accommodate the needs of people with food allergies or intolerances, the task of identifying safe products in the grocery store remains challenging. The problem is not necessarily a *lack* of available information, but rather the feeling of “information overload” that arises from having to interpret endless food declarations in order to assess whether products are safe. Fine print, lack of standardized presentation and presence of unfamiliar terms require substantial cognitive effort and specialized domain knowledge on the customer side.

In this thesis, I have applied insights from the library and information sciences and Semantic Web technologies in order to develop a system meant to support people with food sensitivities in their search for safe foods.

I would like to thank my teachers and fellow students for their input and encouragement during my work with this thesis, and my dear husband and son for their everlasting patience and inspiration.

# TABLE OF CONTENTS

- PREFACE..... 3**
- TABLE OF CONTENTS..... 4**
- 1 INTRODUCTION..... 8**
  - 1.1 INFORMATION NEEDS EXPERIENCED BY CONSUMERS AFFECTED BY FOOD SENSITIVITIES ..... 8
    - 1.1.1 *Food sensitivity prevalence and consequences for affected individuals..... 8*
    - 1.1.2 *Information need in the shopping situation ..... 8*
    - 1.1.3 *Observations from The Norwegian Coeliac Society’s Facebook forum ..... 10*
    - 1.1.4 *Personas illustrating diversity within the customer segment..... 11*
- 2 BACKGROUND ..... 14**
  - 2.1 EXISTING MEASURES AIMED AT HELPING PEOPLE WITH FOOD SENSITIVITIES ..... 14
    - 2.1.1 *Labeling requirements and practices ..... 14*
    - 2.1.2 *Elimination of common allergens in mainstream products..... 18*
    - 2.1.3 *Physical organization of grocery stores..... 20*
    - 2.1.4 *Printed material and “brochure websites” ..... 21*
    - 2.1.5 *Online stores ..... 22*
    - 2.1.6 *Interactive information systems..... 25*
  - 2.2 SHORTCOMINGS OF EXISTING SOLUTIONS – SEEN FROM AN INFORMATION SCIENCE PERSPECTIVE ..... 32
    - 2.2.1 *Use of broad, predefined categories..... 32*
    - 2.2.2 *Suboptimal precision and recall..... 32*
    - 2.2.3 *The fuzzy nature of product safety..... 35*
    - 2.2.4 *Context sensitive relevance perception..... 35*
    - 2.2.5 *Data vs. information..... 36*
    - 2.2.6 *Problems with manual assignment of allergens..... 37*
    - 2.2.7 *Lack of standardization regarding practice of allergen warnings..... 37*
    - 2.2.8 *Vendor-specific sources and systems – lack of industry collaboration..... 38*
    - 2.2.9 *Information loss ..... 38*
    - 2.2.10 *Understanding users’ goals..... 40*
- 3 INTRODUCING AN ALTERNATIVE APPROACH – A DECISION SUPPORT SYSTEM BASED ON SEMANTIC WEB TECHNOLOGIES AND LINKED DATA..... 42**
  - 3.1 REJECTING THE BOOLEAN INTERPRETATION OF PRODUCT SAFETY ..... 42
  - 3.2 DESIGN GOALS ..... 42

3.2.1	<i>Reliable precision and recall performance.....</i>	42
3.2.2	<i>Matching based on semantics rather than text.....</i>	43
3.2.3	<i>Reducing information overload by differentiating between data and information .....</i>	43
3.2.4	<i>Integrating data from multiple industry actors .....</i>	43
3.3	DECISION SUPPORT SYSTEMS .....	44
3.3.1	<i>Definition.....</i>	44
3.3.2	<i>Vast and increasing amounts of data.....</i>	44
3.3.3	<i>Identifying parts that can be left to machines.....</i>	44
3.3.4	<i>User interface .....</i>	45
3.3.5	<i>Core data set administered by a governmental agency .....</i>	45
3.4	A MODEL BASED ON SEMANTIC WEB AND LINKED DATA.....	46
3.4.1	<i>Common domain ontology and authoritative knowledge base.....</i>	46
3.4.2	<i>Automatic inference .....</i>	47
3.4.3	<i>Preprocessing the data .....</i>	47
3.4.4	<i>Traffic light colors for quick feedback.....</i>	48
3.4.5	<i>Smartphone app.....</i>	48
<b>4</b>	<b>PROVIDING A PROOF OF CONCEPT.....</b>	<b>49</b>
4.1	KNOWLEDGE PREREQUISITES AND REFERENCES.....	49
4.2	PROTOTYPE DEVELOPMENT PROCESS .....	50
<b>5</b>	<b>ONTOLOGY DEVELOPMENT .....</b>	<b>51</b>
5.1	THE WEB ONTOLOGY LANGUAGE (OWL) .....	51
5.2	PROTÉGÉ ONTOLOGY EDITOR.....	51
5.3	DATA MODELING .....	51
5.3.1	<i>Classes.....</i>	52
5.3.2	<i>Properties.....</i>	54
5.3.3	<i>Semantic relationships between allergen instances .....</i>	56
5.4	PROBLEMS AND WORKAROUNDS.....	57
5.4.1	<i>Remodeling the relationship between products and ingredients to enable inference...57</i>	
5.5	DISCUSSION AND SUGGESTIONS FOR FUTURE WORK.....	59
5.5.1	<i>Issues with “free from” modeling .....</i>	59
5.5.2	<i>Consequences of semantic relationships between allergen instances .....</i>	60
5.5.3	<i>Semantic relationships between ingredient instances.....</i>	60
5.5.4	<i>Refining the ontology to take into account that products may change over time.....</i>	62
5.5.5	<i>Modeling of binary vs. N-ary relations .....</i>	62
5.5.6	<i>Reuse and mapping to existing vocabularies and ontologies.....</i>	64

<b>6</b>	<b>APPLICATION LAYER DOCUMENTATION .....</b>	<b>66</b>
6.1	IMPLEMENTED USE CASES .....	66
6.1.1	<i>Scan an individual product.....</i>	<i>67</i>
6.1.2	<i>Categorize all products by safety.....</i>	<i>68</i>
6.2	TECHNOLOGY CHOICE.....	69
6.3	OVERALL APPLICATION ARCHITECTURE .....	69
6.4	JSP AND JAVA BEANS.....	70
6.5	UML CLASS DIAGRAM.....	72
6.5.1	<i>The RdfHandler class.....</i>	<i>74</i>
6.5.2	<i>The User class.....</i>	<i>75</i>
6.5.3	<i>The Resource class.....</i>	<i>75</i>
6.5.4	<i>The Allergen class.....</i>	<i>76</i>
6.5.5	<i>The Product class.....</i>	<i>76</i>
6.5.6	<i>The Ingredient class.....</i>	<i>76</i>
6.6	UML SEQUENCE DIAGRAM.....	77
6.6.1	<i>Reading the UML sequence diagram.....</i>	<i>77</i>
6.6.2	<i>Walkthrough of the sequence diagram.....</i>	<i>80</i>
6.7	DISCUSSION AND FUTURE WORK.....	100
6.7.1	<i>Optimizing execution time.....</i>	<i>100</i>
6.7.2	<i>Enabling distributed data.....</i>	<i>100</i>
6.7.3	<i>User testing.....</i>	<i>101</i>
6.7.4	<i>Utilizing the smartphone camera as a barcode scanner.....</i>	<i>101</i>
6.7.5	<i>Serialization of user data.....</i>	<i>101</i>
6.7.6	<i>Refined user profiles.....</i>	<i>102</i>
6.7.7	<i>Open source API.....</i>	<i>103</i>
<b>7</b>	<b>ESTABLISHING THE REQUIRED DATA SETS – DISCUSSION AND FUTURE WORK .....</b>	<b>104</b>
7.1	ESTABLISHING THE AUTHORITATIVE DATA CORE.....	104
7.1.1	<i>Original strategy – semi-automated conversion of existing data source.....</i>	<i>104</i>
7.1.2	<i>Revised strategy – bottom up, frequency-based population.....</i>	<i>105</i>
7.2	SELECTION OF ALLERGENS .....	106
7.3	SEMANTIC DISTINCTION BETWEEN FUNCTIONALLY EQUIVALENT INGREDIENTS.....	106
7.4	DATA QUALITY AND AUTOMATED INFERENCE .....	107
7.5	CRITICAL MASS OF PRODUCT DATA.....	108
7.6	RELIABLE DATA, INFERENCE AND ALGORITHMS.....	108
7.7	USER INTERFACE FOR DATA PROVIDERS.....	108

7.8	FACILITATING EFFICIENT FEEDBACK FROM USERS TO DATA PROVIDERS.....	108
7.9	DEALING WITH UNRELIABLE DATA PROVIDERS.....	109
7.10	RDF-BASED CERTIFICATE SPECIFYING LABELING PRACTICE.....	109
7.11	INCENTIVIZING ACCURATE ALLERGEN LABELING.....	110
7.12	MULTILINGUAL SUPPORT.....	110
7.13	MULTIPLE APPLICATION LAYERS BASED ON THE SAME DATA .....	111
<b>8</b>	<b>CONCLUSION.....</b>	<b>112</b>
<b>9</b>	<b>REFERENCE LIST .....</b>	<b>114</b>
<b>ATTACHMENTS .....</b>		
JAVA SOURCE CODE.....		
	<i>RdfHandler.java</i> .....	
	<i>User.java</i> .....	
	<i>Resource.java</i> .....	
	<i>Allergen.java</i> .....	
	<i>Ingredient.java</i> .....	
	<i>Product.java</i> .....	
JSP AND CSS.....		
	<i>home.jsp</i> .....	
	<i>display_filter.jsp</i> .....	
	<i>available_actions.jsp</i> .....	
	<i>filter.jsp</i> .....	
	<i>categorized_products.jsp</i> .....	
	<i>product_details.jsp</i> .....	
	<i>style.css</i> .....	
OWL ONTOLOGY.....		
	<i>Ontology.owl (manually abridged version)</i> .....	
ONLINE PROTOTYPE AVAILABLE AT <a href="http://ekko.hioa.no:8090/Prototype/">http://ekko.hioa.no:8090/Prototype/</a> .....		

# 1 INTRODUCTION

In this thesis, I start by briefly depicting how food sensitivities lead to information needs in the shopping situation and discuss shortcomings of existing measures, seen from an information science perspective. Based on this, I propose a novel approach based on a Semantic Web and Linked Data. I proceed to elaborate on the development process behind a decision support system prototype that I have developed in order to provide proof of concept for the model.

## 1.1 Information needs experienced by consumers affected by food sensitivities

### 1.1.1 Food sensitivity prevalence and consequences for affected individuals

Many people experience adverse reactions to foods, and the problem appears to be increasing in the population. According to Chief Physician Roald Bolle, about 1.5-2.5% of the adult population and about 5-8% of young kids have reactions that can be demonstrated objectively. Food sensitivities may have many different manifestations, ranging from mild symptoms to life-threatening allergic shock. In many cases, the mechanisms behind the reactions are unclear, thus the exact prevalence of food hypersensitivity is hard to determine. Estimates vary, depending on the definition and methods used. Food *allergies*, in the strict medical sense, only account for a subset of the cases. Conversely, as much as one fourth of the adult population self report that they have experienced some form of adverse reaction towards food. (Bolle, 2012)

Currently, no pharmacological treatments can prevent the problem – the only solution for the affected people is to avoid the substances that cause their symptoms (Bolle, 2012; Bueso, 2012a).

### 1.1.2 Information need in the shopping situation

For people who are affected by food allergies or intolerances – directly or indirectly – grocery shopping can be quite challenging and time consuming. The process of identifying safe products shares many characteristics with information seeking. The consumer experiences a



“knowledge gap”, and may engage in several different activities in order to satisfy her information need:

- Seeking out the specialized “free from” shelves to increase the likelihood of finding relevant products
- Consulting information material such as printed brochures and company web sites
- Posting questions at online forums or social networking sites, such as Facebook
- Picking out desirable products and evaluating them based on their product declarations
- Falling back on a limited set of products already known to be safe

In order to identify safe products, a consumer may need substantial knowledge about what ingredients are safe for her to eat and not. Several different ingredients may pose a risk, so it is not enough to quickly skim read the content declaration, looking for the exact terms that represent the substance one is allergic to. Figure 1 illustrates how a person who is allergic to milk has to look out for a large set of ingredients, some of which are less obvious. It’s easy to see how this can create a feeling of “information overload”.

**Figure 1** Word cloud illustrating the variety of terms that may appear in a product declaration representing a common allergen such as “Milk”.



A large number of people are *indirectly* affected by the problems that arise from conditions

such as food allergies, coeliac disease and severe cases of food intolerance – as parents, grandparents and people who prepare foods for others as part of their job.

The described customer segment is extremely diverse, ranging from people with severe allergic reactions such as anaphylactic shock, to people who experience mild discomfort after ingesting too much of a particular substance. Some have allergies diagnosed by medical doctors. Others go through rigorous testing without finding any explanation for the symptoms they experience, but still decide to avoid certain substances because they believe that it makes them feel better.

The customer segment is also diverse in the sense that people may be allergic or intolerant to a wide specter of substances, and combinations of these. It is therefore hard to predict individual customers' needs.

For actors in the food industry, it is irrelevant whether a customer's dietary need is formally diagnosed or perceived – it is in their commercial interest to accommodate all their customers in the best possible way.

### **1.1.3 Observations from The Norwegian Coeliac Society's Facebook forum**

As part of my exploration of this domain, I have been regularly following The Norwegian Coeliac Society's (Cøliakiforeningen) forum on Facebook. The forum has just over 3.500 members as of January 2013, and is just one out of many social networks available for people with food sensitivities.

I started following the forum because I thought it would be a valuable source of insight into the situation that people with food sensitivities find themselves in, and the information needs they experience. Without performing any systematic, in-depth research about the use of this forum, I have observed some patterns and recurring request types. The users appear to be a mix of people that either have coeliac disease themselves, or have kids or other family members with the diagnoses. Some have had their diagnosis for years, while others are newly diagnosed or still in the process of verifying the condition. Some have even tested negative

for coeliac disease, but still feel the need to avoid gluten in their diet, and thus find the forum a useful source of information and knowledge exchange.

There are many different types of posts every day, spanning from people asking for advice on places to eat in a foreign country they are about to visit, to tips about gluten free products and recipes. One recurring request type of special interest to my project follows the pattern: “Is it safe for us to eat X?” First of all, it is interesting to observe that in spite of product declarations readily available, some people still prefer to ask their forum peers for a second opinion.

It seems that quite a few people are uncertain about which ingredients represent a risk, and thus find the food declarations hard to interpret. Secondly, the answers they get upon posting a request like this uncover one basic fact; in spite of having the same medical diagnosis, people seem to have very different tolerance levels and thus different practices when it comes to evaluating the safety of foods.

There is no formal moderation of the forum, so any member is free to share his or her advice, anecdotal evidence and opinions, often leading to long discussions between forum members. Instead of reaching a consensus, these discussions tend to culminate with a shared understanding that people have different needs, even if they share the same diagnosis. In the end, every person has to find out what works for him or her.

#### **1.1.4 Personas illustrating diversity within the customer segment**

The following personas are meant to illustrate the diverse needs of people with various food sensitivities. Persona A and B are inspired by my observations from NCF’s Facebook forum. Persona C, D and E exemplify people with food allergies and intolerances.

	<b>Persona A</b>	<b>Persona B</b>	<b>Persona C</b>	<b>Persona D</b>	<b>Persona E</b>
<b>Gender, age, language proficiency</b>	Female, 35 years old, Norwegian native speaker.	Female, 22 years old, Italian exchange student with very limited Norwegian skills.	Male, 43 years old, Norwegian native speaker.	Female, 70 years old, Norwegian native speaker.	Male, 15 years old, Norwegian native speaker.
<b>Background, "problematic situation"</b>	Mother of a newly diagnosed coeliac toddler. The child is lactose intolerant due to temporary damages to the intestinal wall. Doctors have routinely advised to avoid oats in the child's diet for the first 6 months. The parents thus need to double-check all products, even if labeled "gluten free". Has been told that trace amounts of gluten are OK, but is otherwise unwilling to take any risks on the child's behalf.	Experienced coeliac, possible also allergic to wheat. Extremely sensitive, needs to avoid all ingredients derived from gluten-containing grains (even if the ingredients' gluten content is below the threshold set to label a substance "gluten free") and all trace amounts from production environments. Needs to double-check all products, even if they are labeled "gluten free". May choose to eat something that is labeled with the warning "may contain traces of wheat" on rare occasions, if no other options are available and she doesn't have a "big day" coming up.	About to arrange a birthday party for his seven year old son. Has been informed in a PTA meeting that three of the classmates suffer from food allergies or intolerances. The kids are too young to give reliable accounts about what foods they can or cannot eat.	Grandmother of a child who has been suffering from bloating and stomachaches for some time. Several tests have been carried out with negative results. The MD suspects that the symptoms may be caused by some form of food intolerance. It has therefore been decided that lactose should be eliminated from the girl's diet for a trial period.	Severe allergy towards eggs, fish, milk and nuts. Especially sensitive to nuts during the pollen-season due to cross-sensitization. Wants to be independent and hang out with his friends without having to worry about access to safe food. His parents worry that he doesn't always bother to double-check foods when he is away from home.
<b>Dietary restrictions</b>	Gluten, oats, lactose.	Gluten (incl. trace amounts), wheat (incl. trace amounts).	Milk, eggs, peanuts.	Lactose.	Eggs (incl. trace amounts), fish (incl. trace amounts), milk and nuts.
<b>Preferences</b>	Looks for child friendly food, such as fish cakes or meatballs with pasta and sauce, which the whole family can enjoy together.	On a budget. Likes to cook "from scratch". Tired of the typical "free from" products. Willing to experiment with new ingredients. Wants healthy food.	Eager to find a meal that everyone can enjoy without too much fuss. Looks for semi-finished products such as hotdogs and cake mix where milk, eggs and peanuts have been eliminated. Wants to "play it safe" by completely avoiding foods that may cause reactions in some of the kids.	Wants to continue using her own traditional recipes and favorite products. Looks for satisfactory replacements for key ingredients such as butter and double cream and her grand kids' favorite treats.	Doesn't want to stick out. Prefers to eat the same type of food as his mates or not at all.

	<b>Persona A</b>	<b>Persona B</b>	<b>Persona C</b>	<b>Persona D</b>	<b>Persona E</b>
<b>Knowledge and experience base, "knowledge gap"</b>	Unfamiliar with which ingredients are safe or not for her child. Insecure about product choices and finds little or no help in the store. Uses vendors' brochures aimed at people with food sensitivities and the Coeliac Society's magazine as main sources of product information.	Substantial knowledge about food and allergens in own language. Unfamiliar with Norwegian terminology.	No personal experience with food allergies or intolerances. Has been given some basic advice in a PTA-meeting.	Substantial knowledge about traditional food and cooking. Not aware of "hidden allergens" in industrial products.	Lacks sufficient knowledge and patience to recognize all risk factors.
<b>Additional challenges affecting search behavior</b>	Finds it hard to concentrate on interpreting the labels when she shops with her toddler.	Insufficient Norwegian skills to "decode" all the ingredient terms occurring in the product declarations.	Pressured for time. Lacks incentive to acquire in-depth knowledge about food allergies and intolerances, as it doesn't affect him on a regular basis.	Has trouble reading the fine print used on product declarations. Struggles a bit with short-time memory.	Gets extremely bored with reading product declarations. Possibly dyslectic. In denial about the whole problem.
<b>Consequences</b>	"Plays it safe" by choosing the same products over and over again.	Sticks to brands that she's familiar with from home or other imported products with multilingual declarations.	Considers calling the parents of the affected children, asking them to bring their own safe food.	Has inadvertently given the granddaughter candy without checking for occurrence of lactose. Extremely cautious due to earlier mistakes. Eliminates far more foods than strictly necessary, affecting the whole family's joy of food.	Mishaps happen a lot, making him ill for days. Ends up eating the same products over and over again. His parents are worried about the lack of variation in his diet.
<b>Smartphone access and mastery</b>	Owens an iPhone, but doesn't have time to use it while grocery shopping.	Owens an Android phone. Uses Google to find information about unfamiliar ingredients and products while in the store, but finds it time consuming to go through the information.	Owens an iPhone. Expert user. Always online, "googling" or using various "apps" in order to solve tasks.	Owens an iPhone, but rarely uses any advanced functions. Her kids has installed a few apps and taught her how to use them. She is willing to use the ones that she immediately recognizes the value of (e.g. a weather forecast app).	Owens a new Android phone and is extremely eager to use it.

## 2 BACKGROUND

### 2.1 Existing measures aimed at helping people with food sensitivities

In this chapter, I'll briefly depict some of the ways in which different actors in the food industry are currently attempting to aid users in their search for safe products.

#### 2.1.1 Labeling requirements and practices

The only available treatment for a person with food hypersensitivity is to avoid the foods that he or she is sensitive to. Food declarations provide consumers with vital information and are the only utility available for determining whether products contain substances that cause adverse reactions. Labeling is especially important for composite industrial products where common sense falls short and consumers may be oblivious to “hidden” allergens. Companies that manufacture or import foods are responsible for ensuring that their food is labeled according to current legislation. (Bueso, 2012b).

##### 2.1.1.1 EU's common regulatory framework

Not being a member of the European Union, Norway largely follows EU's directives regarding food labeling due to the EEA agreement. (Løvik, 2012a). Harmonized regulations facilitate free flow of goods between member states and EEA countries while ensuring a high level of food safety for consumers (Mattilsynet, 2012).

##### 2.1.1.2 Regulatory development process

The regulatory development is a complex process engaging multiple stakeholders such as business associations, patient organizations, scientists and policymakers. In the EU system, *risk assessment* and *risk management* are separate processes that are performed by different entities. A panel of independent experts subordinated *The European Food Safety Authority* (EFSA) is responsible for the *risk assessment* based on thorough analysis of current scientific documentation. The *European Commission* is responsible for the *risk management*, i.e. identifying and enforcing measures to reduce the risks. Having to balance several different

interests in a pragmatic way, The European Commission's regulations largely reflect EFSA's advice. (Løvik, 2012a).

### *2.1.1.3 Current labeling requirements*

In Norway, the Food Act with regulations implement the current EU-directives regarding food labeling (Matloven, 2003; Merkeforskriften, 1993; Næringsmiddelhygieneforskriften, 2008).

Matportalen.no, an online portal to quality information about food and health from the public authorities in Norway, sums up the regulations as follows:

The labeling shall include information that enables consumers to avoid foods they cannot tolerate.

According to the regulations, all packaged food must be labeled with specific information about the product, such as product description and ingredient list. The ingredient list should include all ingredients that occur in the product. Some exceptions exist, but do not apply to allergenic ingredients.

Some ingredients known to cause allergy and hypersensitivity reactions are subject to particular labeling requirements. These ingredients should always be declared in the ingredient list on the package with a clear reference to the raw material . . . .

Allergenic ingredients with special labeling requirements are gluten-containing grains, fish, crustaceans, molluscs, eggs, peanuts, lupine, soy, milk (including lactose), nuts, celery, mustard, sesame and sulfites. Products made from these foods should also be labeled with allergenic ingredients.

New labeling regulations that will be effective from December 2014, require that ingredients that may cause allergies or intolerances should be highlighted in the labeling. This can be done using for example bold or italic, for increased visibility. Some products are already labeled this way.

(Mattilsynet, 2013, own translation).

In The Norwegian Health Library's special issue on food allergy, food intolerance and other hypersensitivity reactions to food, Bueso (2012a) exemplifies how the labeling requirements should be practiced. Each ingredient should be labeled in a way that is understood by the consumer. Foreign phrases such as “couscous” or “tahini” should be replaced by more familiar terms from the national language. The term “nuts” is too general and should be replaced by a more specific term (e.g. walnut, pistachio or almond). When listing gluten-containing ingredients, it should be made clear which specific grain they originate from (e.g. wheat, barley or rye). Food-additives derived from allergenic substances should also state the origin, e.g. “E322 (soy lecithin)”. All ingredients derived from allergenic substances should be declared both by their own name and the name of the substance they are derived from. Ingredients not included in the list of allergens mentioned above, should be declared if they constitute more than 2% of the product’s total weight. Some foods contain composite ingredients such as margarine, jam, mayonnaise, mustard or marzipan. In these cases, the name of the composite ingredient should be followed by a specification of the components. General terms such as “spices” or “breadcrumbs” are allowed, but allergenic components should be specified in brackets, e.g. “breadcrumbs (with wheat, eggs and soy)” or “spices (celery)”.

Products aimed at people with coeliac disease or gluten-intolerance are subject to additional regulations (Forskrift om glutenfrie varer, 2009). The label “gluten-free” should only be used on products that contain less than 20 mg gluten per kilo, whereas the label “very low gluten content” should only be used on products that contain less than 100 mg gluten per kilo. Gluten-free products can either be products where the gluten has been removed in order to



satisfy the aforementioned requirements or products without gluten-containing ingredients where special measures have been taken to avoid contamination.

For comprehensive information about the current labeling requirements, please refer to *Directive 2000/13/EC of the European Parliament and of the Council of 20 March 2000 on the approximation of the laws of the Member States relating to the labeling, presentation and advertising of foodstuffs* (European Parliament, Council, 2000). Allergens that are subject to special requirements are listed in *ANNEX IIIa*.

#### *2.1.1.4 Voluntary labeling of contaminants*

The aforementioned regulations ensure that products are labeled according to their *recipe*. However, the production environment may cause products to be *contaminated* by allergens during the manufacturing process because several different foods are produced using the same production facilities. In such cases, the allergens won't be reflected in the ingredient list.

Food manufacturers are required to provide documented procedures that address both hygiene requirements and appropriate labeling (Mattilsynet 2012; Næringsmiddelhygieneforskriften, 2008; Internkontrollforskriften for næringsmidler, 1994). Allergenic substances should be labeled regardless of the amount.

Many products are currently labeled with warnings such as “May contain traces of X,Y and Z”. Food manufacturers use these warnings as a means of informing consumers about possible allergenic contaminants. This is a *voluntary* labeling practice and there is no formal definition declaring what a “trace” amounts to or how frequently the allergenic substance should occur in order for a product to be assigned such a warning. (Mattilsynet, 2012)

The Norwegian Food Safety Authority (Mattilsynet) has investigated the use of such warnings as part of a larger Nordic study (Mattilsynet, 2012). According to the study, many different formulations are currently in use, such as “may contain traces of ...”, “may contain ...”, “produced on a production line that also produces products containing ...” and “produced in a facility that also produces products containing ...”. Some manufacturers instead refer consumers to specific websites for more information about allergens. Others

explicitly warn about *all* allergens occurring in the product, even if the allergens are already evident from the ingredient list. The study uncovered highly variable labeling practices across the industry, ranging from omission of allergenic substances in the ingredient list to excessive use of allergen warnings without proper cause. (Mattilsynet, 2012).

The Norwegian Food Safety Authority notes that food producers should take adequate measures to prevent unintentional presence of allergens in foods. Both manufacturers and importers should have knowledge about the risks and take precautions to minimize it. Labeling should not be used as an excuse for poor control and hygiene management. (Mattilsynet, 2012).

Unfounded warnings may unnecessarily constrict the range of products available to people with food sensitivities and in the long run decrease consumers' trust in allergen labeling altogether. Allergen warnings should thus only be used when it is absolutely necessary. (Mattilsynet, 2012). Løvik (2012b) points out that excessive labeling may even have the adverse effect of making consumers less attentive towards labels, causing them to miss out on or disregard essential information.

### **2.1.2 Elimination of common allergens in mainstream products**

Vendors increasingly offer “mainstream” products where some of the most common allergens have been eliminated. In a press release from 2012, NHO Mat og drikke (a subdivision of The Confederation of Norwegian Enterprise concerned with the food industry) describes the current trend: “There is a market out there for food free of allergens. The industry has adapted to consumers' needs and desires. More and more produce food for allergy sufferers.” (NHO Mat og drikke, 2012, own translation).

#### *Example: Toro*

TORO is one of the largest and best-known food brands in Norway, with more than 750 individual products on the market (Toro, n.d.). The aforementioned press release describes Toro as an industry pioneer: “Toro has gone one step further, thinking that every new product they develop should suit everyone, including people with allergies.” (NHO Mat og drikke, 2012, own translation).

It is, however, conceivable that emphasizing the fact that a mainstream product is free from common allergens may put other customers off, believing that that the taste or texture may be inferior to alternative products. It appears that vendors sometimes choose to “downplay” the fact that their mainstream products are suitable for people with food sensitivities in marketing efforts aimed at the general population. Vendors like Toro reach people affected by food sensitivities by targeted marketing through patient organizations, printed brochures listing products free from specific allergens, interactive online services and “word of mouth” in social media.

*Example: Coop*

Coop is one of the major supermarket operators in Norway and currently runs 840 supermarkets. The following is an excerpt from a recent press release:

...Coop’s goal is that as many of their own brand products as possible should be available to all customers. Therefore, the grocery operator continuously strives to remove allergens from their products. This fall, all Coop’s own sausages were made guaranteed free from gluten and lactose.

Now Coop has eliminated gluten from all breaded pork products and changed the production of these to clean manufacturing facilities. These products are thus 100 percent gluten-free. The products are labeled “Mat UTEN gluten” (Eng.:“Food WITHOUT gluten”) in order to inform the customers.

–With us, customers should be able to shop for all the family. We want to give families living with allergies and intolerance a simpler life. We know that breaded products are popular everyday food, especially among young people, so it is important

to make these available to more people, says Hege Berg-Knutsen, who is responsible for Coop's own brands. . . .

–Another positive effect of the change to gluten free production is that we can now guarantee that many of our other processed meat products . . . are 100 percent gluten-free. Our customers will therefore experience greater flexibility and more options when they shop in our stores, says Berg-Knutsen. . . .

(Coop, 2013, own translation).

For people with allergies, it is a positive trend that more vendors are taking their needs into account by producing food that most people can eat. However, since these products are distributed throughout the grocery store, the problem of *identifying* the products remains.

### **2.1.3 Physical organization of grocery stores**

Most grocery stores accommodate people with food allergies and intolerances by offering separate shelves reserved for products free from common allergens. In practice, this helps these users to narrow down the “search” to a subset of products that are more likely to be safe. There are several reasons why this approach, by itself, is suboptimal.

Firstly, most people with allergies are able to eat products from several other parts of the store, such as fresh produce, refrigerated and frozen goods and canned goods. Referring all these people to the “free from” shelves may create the impression that the remainder of the store is “off limits” – which is fortunately not the case. This approach may thus contribute to an unnecessary limitation the customer’s choice, since he or she may be unaware of all the other safe products that are placed elsewhere in the store, or just put off by the thought of investigating each individual product to judge whether it’s safe to eat or not.

Figure 2 Example of "free from" shelves at the high-end grocery store Meny<sup>1</sup>



Secondly, the “free from” shelves are often placed next to other niche products, such as organic, vegetarian, “low-carb” and so-called “super-food” products. This may create confusion among inexperienced customers and entails a risk of selecting unsafe products, such as spelt (which is a special type of wheat) instead of gluten-free flour.

Lastly, commonly used categories such as “milk-free” or “gluten-free”, are insufficient to help people with multiple or atypical allergies or intolerances. They will still need to double-check each individual product declaration.

It is in everyone’s interest that all customers gain access to *all* products that they can safely eat, and the “free from”-shelves offers limited support in this regard.

#### 2.1.4 Printed material and “brochure websites”

Many different actors, ranging from governmental agencies to patient interest groups to vendors and shops offer digital and/or printed material specifically aimed at people with food allergies and intolerances and their friends and family. Such resources can be very helpful for newly diagnosed patients who want to learn more about their own diagnosis or consumers who e.g. want a complete list of all gluten-free products from a particular vendor. However, factors such as format (ranging from verbose texts to spreadsheets), ties to specific vendors and/or lack of customization may limit the usefulness of such resources during the actual

---

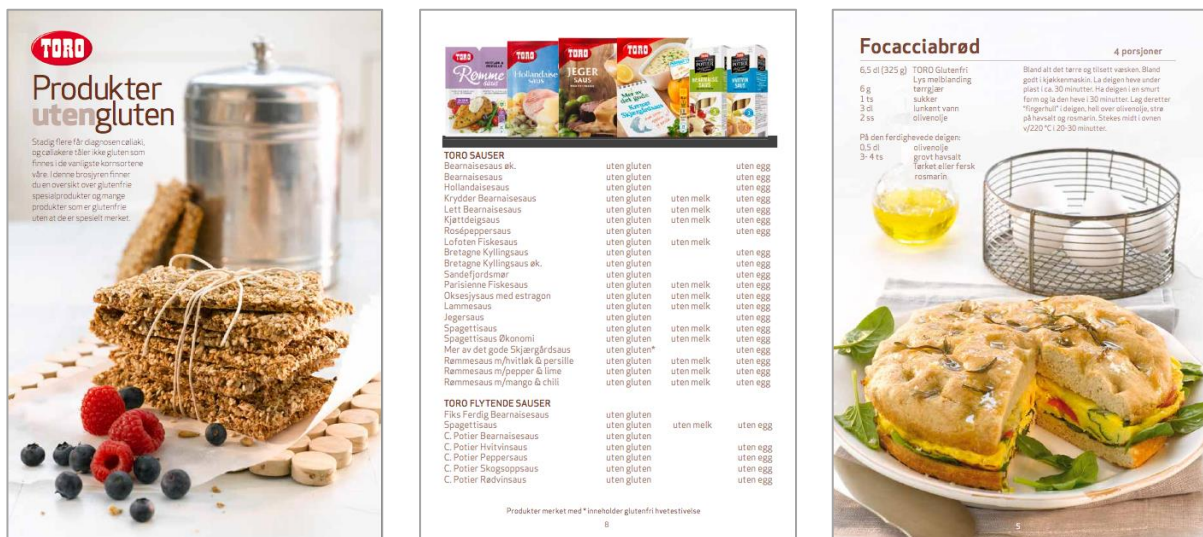
<sup>1</sup> The image is copied from <http://www.meny.no/Info/Miljo-og-helse/Allergi/>

shopping situation. Web pages containing spreadsheets or scripts may be ill combined with smartphones. Printed material, on the other hand, easily gets outdated.

*Example: Toro's brochure about products free from gluten*

This brochure provides consumers with a comprehensive list of Toro's gluten-free products. For each product group, a simple table indicates which products are free from gluten, wheat starch, milk and eggs. Moreover, small pictures of the products help users know what to look for in the store.

**Figure 3 Excerpt of Toro's brochure of products free from gluten**



This approach is very helpful, but only directs customers towards products from a particular vendor.

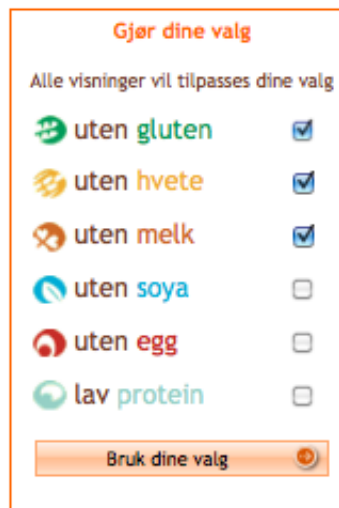
**2.1.5 Online stores**

Some online stores allow the customer to filter products by allergens. One problem with this solution is that it assumes a finite set of allergies. This is not the case in real life, as people may be over-sensitive towards virtually any substance. Also, it appears that these solutions are based on manual assignment of allergen information to each product. Again, this approach does not suit people with atypical dietary needs. The manual effort involved leads to scalability-problems and makes solutions like these prone to human error.

*Example: Allergimat.no (Eng. trans.: Allergy foods)*

Allergimat.no is an online grocery store directed at people with PKU (Phenylketonuria)<sup>2</sup>, coeliac disease and food allergies<sup>3</sup>. On the welcome page, users are presented with a simple form where they can specify their dietary needs. The available options only partially overlap with the allergens that are subject to mandatory labeling. Allergens like nuts, sesame, crustaceans, mollusks and fish are left out, while wheat and low-protein (for PKU patients) are added to meet customers' needs.

**Figure 4 Excerpt of screen shot showing substances that can be eliminated at Allergimat.no**



Gjør dine valg	
Alle visninger vil tilpasses dine valg	
<input checked="" type="checkbox"/> uten gluten	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/> uten hvetete	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/> uten melk	<input checked="" type="checkbox"/>
<input type="checkbox"/> uten soya	<input type="checkbox"/>
<input type="checkbox"/> uten egg	<input type="checkbox"/>
<input type="checkbox"/> lav protein	<input type="checkbox"/>
<input type="button" value="Bruk dine valg"/>	

Once the user has submitted the form, the website “remembers” the settings for the remainder of the session and when the user returns on later occasions. Only foods that are in line with the specified dietary needs are presented to the user when browsing or searching for foods. The presentation of individual products contains a field specifying that the product is manufactured without some specified allergens and/or is low-protein. These “facets” appear to have been assigned manually and constitute the basis for the refinement of presented products according to the user’s settings.

Manual assignment of data about products’ suitability for special dietary needs requires specialized expertise and accuracy. This approach is adequate in the case of Allergimat.no, because the store specializes in dietetic food and thus has in-depth insights into their customer

<sup>2</sup> PKU (Phenylketonuria) is a rare metabolic disease that is treated with a lifelong low protein diet (Senter for sjeldene sykdommer, 2012).

<sup>3</sup> The service was tested in May 2013.

segments' dietary needs. For ordinary grocery stores with far greater range of products, this approach would probably not suffice, because of scalability issues and risk of human error.

The limited set of allergens available for selection means that some users may be unable to adequately express their needs to the system. Persona A would for instance need to double-check all gluten free products for any occurrence of oats, because oats is not provided as an option.

Figure 5 Screen shot from Allergimat.no showing example of a gluten-free product that contains oats

The screenshot shows a web browser window displaying the Allergimat.no website. The browser's address bar shows the URL: <https://www.allergimat.no/butikk/s/products/view.html?p...>. The website header features the Allergimat.no logo and a search bar. Below the header is a navigation menu with categories: Produkter, Oppskrifter, Nye produkter, Tilbud & Tips, Produkter A - Å, Linker, and PKU. The main content area is divided into three columns. The left column contains a 'Dine valg' section and a vertical list of product categories, with 'Erstatning for melk' highlighted. The middle column displays the product 'Carlshamn Godt i Maten' with a price of kr. 19.50 and a 'Legg i handlekurv' button. The right column shows a 'Handlekurv' section indicating it is empty, 'Leveringsinfo', and 'Nyttige tips'. The product details include: 'Anvendelse: Kan ikke piskes.', 'Pr. 100g: Energi 670 kJ / 160 kcal., Protein 0,8 g, karbohydrater 6,0 g (herav sukkerarter 5,2 g), fett 15 g (herav mettet fett 1,2 g), kostfiber < 1,0 g, natrium 0,09 g, fenylalanin 40 mg.', 'Varen er fremstilt uten: Gluten, hvete, melk, soya, egg og er lavprotein.', 'Oppbevaring: I romtemperatur når uåpnet. Holdbar ca 5 døgn i kjøleskap etter åpning.', 'Innhold: Vann, rapsolje, havreflåk (10%), emulgator (E472e), surhetsregulerende middel (E331 og E330), fortykningsmiddel (E415).', 'Leverandør: Carlshamn meieri', and 'Varenummer: 751705'.



## 2.1.6 Interactive information systems

### 2.1.6.1 Dynamic websites

Some vendors use dynamic web sites as a means to help users with special dietary needs.

*Example: Gilde's product search.*

Gilde is one of the largest suppliers of meat products in Norway. In an attempt to accommodate people with food sensitivities, the company offers an online product search on their web page<sup>4</sup>. The search form contains a search box and a checklist of 14 substances that are known to cause allergic reactions. All products containing the checked substances are filtered out from the result set. The user is required to provide a search string of at least three letters in order for the search to be carried out. This makes the service unable to provide the user with a complete overview of safe products.

Again, the filter is based on a finite list of allergenic substances. The list contains all the allergens from Annex IIIa, with the addition of *paprika*.

**Figure 6** Excerpt of screen shot of Gilde's product search

The screenshot shows a web interface for product search. At the top, there are three tabs: 'Produktsøk' (selected), 'Oppskriftsøk', and 'Utvidet oppskriftsøk'. Below the tabs is a search box containing 'leverpostei' and a button labeled '» Skjul Allergisøk'. Underneath is a section titled 'Ikke vis produkter som inneholder:' with a grid of 14 allergen checkboxes. The checked items are 'Gluten' and 'Melkeprot. inkl laktose'. Other allergens listed include Fisk, Nøtter, Selleri, Sennep, Sesamfrø, Paprika, Lupin, and Bløtdyr. A note below the checkboxes states: 'Når ingen av disse ingrediensene er huket av søkes det i alle produkter'. A blue 'Søk' button is located at the bottom right of this section. Below the search results, it says 'Resultater for søk etter "leverpostei". Antall treff 2. Viser treff 1-2.' Two product listings are shown, each with a small image of the product and a '» Les mer' link.

<sup>4</sup> The service was tested July 11, 2012.

In the example shown in Figure 6, I made a search for products without gluten and milk products, corresponding to the typical needs of a person with celiac disease, like Persona A's child.

However, both the products in the result set contain *wheat starch*, which a subset of people with celiac disease, like Persona B, need to avoid due to trace amounts of gluten. This group of people will still need to manually evaluate the ingredient list of each product. The limited amount of available categories leads to low precision, because the user is unable to communicate her actual needs to the search engine. This may lead to purchasing patterns where customers with special needs end up buying the same products over and over again, to save time and effort.

In the list of substances that can be filtered out from the result set, all milk protein and lactose are merged into one category. Newly diagnosed coeliacs like Persona A's child, are often temporarily lactose intolerant, but still able to ingest milk protein. This means that some people may end up with an unnecessary narrow result set due to over-generalization. In information retrieval terms, this corresponds to low recall.

*Example: Matvareguiden.no (Eng. trans.: The food guide)*

Matvareguiden.no is a privately run, ad-funded website with food-related information<sup>5</sup>. The website conveys detailed information about a wide range of products across multiple vendors. Many different features are offered, and for the purpose of this thesis, I have briefly tested the search function aimed at people with allergies or other dietary restrictions.

The search form contains several dropdown menus where the user may select a main category, a sub category, a particular vendor, a main group, a dietary restriction or allergy, a search string or any combination of these criterions. The main categories available are *beverages, health foods, foods and food supplements*. I find it a bit confusing that the sub categories presented in the form is a conglomerate of sub categories belonging to all of the mentioned main categories, regardless of what main category has already been selected. The search function requires that all of the criteria specified by the user be satisfied (i.e. there is an

---

<sup>5</sup> The service was tested May 7, 2013.

implicit AND operator between the criteria). Several combinations of main categories and sub categories thus lead to zero hits.

The dietary restrictions the user may choose from range from allergens subject to mandatory labeling requirements to *fair trade, ecological, vegetarian, halal, low-carb, sugar free, fat free* and *yeast free* products. However, it is not possible to select more than one dietary restriction in the same search, so people with multiple food sensitivities will need to double check the results manually. Product entries reveal that the underlying data model allows products to be assigned more than one dietary restriction (e.g. gluten free and milk free). This shows that the problem arises from the implementation of the search function (or the design of the user interface) and not the data model.

I carried out some searches for products from a specific vendor (Finax), alternating the dietary restriction between “milk free”, “lactose free” and “without milk protein” in turn. I wanted to see if products that were defined as milk free would also turn up in the result sets of the more specific searches. Unfortunately, I found examples of the opposite (see Figure 7 - Figure 10). This problem is due to lack of semantic relationships between the set of dietary restrictions that the user may choose from. Eliminating this problem would require changes in both the data model and application layer.

Users are notified in the search form that only products with asserted allergy or dietary properties are included in the result set. This implies that the result will leave out many products that would have been safe for the user because the products were not labeled specifically as “free from” the particular allergen.



### *2.1.6.2 Smartphone applications (“apps”)*

Norway has very high prevalence of smartphones and availability of wireless Internet access. The international market for smartphone applications is huge, and many different actors develop “apps” for all kinds of purposes. A full review of the apps targeted at people with food sensitivities is beyond the scope of this thesis. However, I have chosen to present one example that stands out, being developed here in Norway in collaboration with major patient organizations.

#### *Example: CheckContent*

CheckContent is a smartphone application available for both iPhone and Android phones. The app offers various features meant to help users shop and cook for people with food allergies, intolerances or coeliac disease. Experts representing the Norwegian Asthma and Allergy Association (NAAF) and the Norwegian Coeliac Society (NCF) perform quality assurance of the underlying data. (NAAF, 2013; CheckContent, n.d.)

The app lets the user specify which allergens he or she wants to be warned about. The available options are consistent with the list of allergens that are subject to mandatory labeling. The user may proceed to scan product barcodes or QR-codes in the grocery store in order to get immediate feedback about whether the product contains any of the specified allergens. The app utilizes the smartphone’s built-in camera as a scanner.

The first time I tried the app, I was a bit confused by the user interface. Upon scanning a product, some basic information was displayed, followed by a list of the allergens I had selected, paired with green “check” signs. I was uncertain whether to interpret the check sign as “Yes, we found this allergen in the product” or “Yes, the product is cleared”. However, after scanning a couple of products, the meaning became obvious after seeing examples of red caution triangles used to communicate that a specified allergen was found in the product.

The screen shots in Figure 11 illustrate scanning of a liver pate from Gilde after checking gluten and nuts in the settings<sup>6</sup>. The app displays a red warning triangle for gluten. This conflicts with the product declaration provided on the package and on the vendor’s website (Gilde, n.d.). Gilde specifically asserts that all of the product’s ingredients are gluten free. The

---

<sup>6</sup> The service was tested May 9, 2013.

ingredient list includes wheat starch, but this is allowed under the current regulations for gluten free food.

Figure 11 Screen shots from CheckContent for iPhone



In the CheckContent app, users may click on the name of an allergen, either in the settings or in the product view, to read a brief note explaining the criteria used to interpret a product as free from the given allergen. In the case of gluten, it is stated that wheat starch is regarded as gluten free. Even if this would cause problems for extremely sensitive people like Persona B, this practice is to be expected due to the current labeling regulations. However, I was surprised to learn that CheckContent also regards products that “may contain traces of gluten” as gluten free. My impression from the Coeliac Society’s Facebook forum is that many coeliacs need to stay clear of trace amounts.

According to the Norwegian Coeliac Society, CheckContent uses the following practice for indicating gluten content in products:

- A green check symbol: the product contains less than 20 ppm gluten (gluten free)
- An unfilled red warning triangle: the product contains between 20-100 ppm (very low gluten content).
- A solid red warning triangle: the product contains gluten and should not be eaten by coeliacs.

(NCF, 2013, own translation).

I did not find a similar explanation of the practice within the application or on CheckContent's own website.

According to CheckContent's specification of what constitutes a gluten free product, the gluten warning assigned to the liver pate is definitely misleading.

For each product, the interface specifies whether the manufacturer has supplied information to CheckContent and whether the product declaration has been reviewed by NAAF or NCF. Judging from my own limited testing it appears as though few vendors have disclosed their data to CheckContent at this point.

The CheckContent app provides several functions and social features. Among other things, users are in some cases referred from an unsafe product to a safe alternative within the same product group.

CheckContent strongly encourages users to manually double check products because vendors may have changed recipes after their products were evaluated.

## 2.2 Shortcomings of existing solutions – seen from an information science perspective

### 2.2.1 Use of broad, predefined categories

Both physical organization of grocery stores and information systems aimed at this group tend to rely on broad, predefined categories, such as “milk-free” and “gluten-free”. In practice, people have unique needs based on what combination of substances they need to avoid, how sensitive they are to trace amounts etc. This means that any solution relying on broad, predefined categories is of limited value.

### 2.2.2 Suboptimal precision and recall

Information retrieval (IR) systems are traditionally evaluated by their ability to return relevant items. Lancaster defines two of the key measurements as follows:

**Precision** – The extent to which the items retrieved in a search of a database are considered relevant or pertinent. A search achieving high precision will be one in which most, if not all, of the items retrieved are judged relevant or pertinent. The precise ratio, a measure of the extent to which precision is achieved, is the number of relevant (or pertinent) items retrieved divided by the total number of items retrieved.

**Recall** – The extent to which all the items in a database that are considered as relevant or pertinent are retrieved in a search of that database. A “high recall” search will be one in which most, if not all, of the relevant (pertinent) items are retrieved. The recall ratio, a measure of the extent to which the retrieval of relevant (pertinent) items occurs, is the number of relevant (pertinent) items retrieved divided by the total number of relevant (pertinent) items in the database.

(Lancaster, 1991, Appendix 3, pp. 291-292)

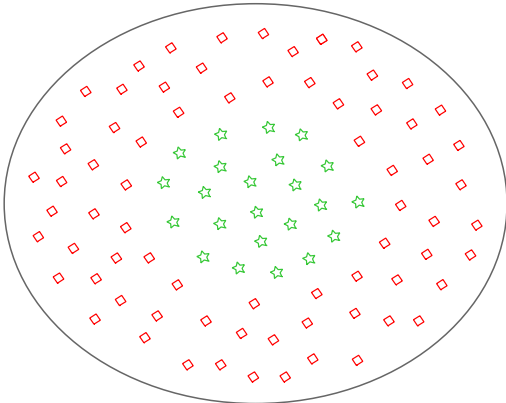


By repeating precision and recall calculations over a large set of search queries and respective relevance assessments made in advance by an independent party, one gets a good indication of a system’s performance. Precision and recall measurements may be combined in various ways to generate a score used to compare different systems.

An ideal system would retrieve all of the relevant documents, and none of the irrelevant ones. In practice, most IR systems score relatively low on both. If modifications are done in order to increase a system’s recall, precision is likely to suffer – and vice versa (Lancaster, 1991, pp. 4). The two performance measures need to be weighed against each other for each IR-system being developed. For some applications, few but pertinent items are preferable. For other applications, a certain degree of noise is an acceptable price to pay to increase the likelihood that all relevant items are recalled.

The purpose of traditional IR systems is to distinguish between *relevant* and *irrelevant* documents in relation to specific users’ information needs. Similarly, the purpose of information systems aimed at helping people with food sensitivities in the shopping situation is to distinguish between *safe* and *unsafe* products.

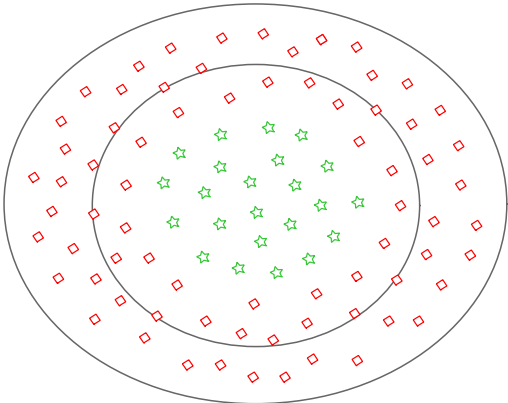
**Figure 12 “Corpus” of products. Green stars represent safe products; red squares represent unsafe products, for a given user.**



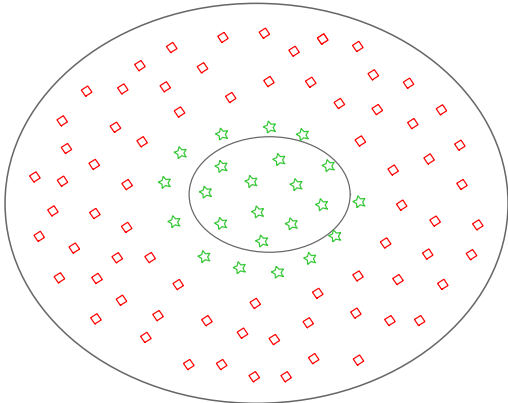
In this context, erroneously presenting unsafe products as safe is intolerable, due to the serious consequences of misguiding people with severe food allergies in their food choices.

On the other hand, presenting safe products as unsafe needlessly constraints the available product selection for people who already have a limited choice.

**Figure 13 High recall, low precision. The result set contains all safe products, but also includes some unsafe products. In many search systems, some degree of noise would be tolerable. In this situation, however, it would be unacceptable, as a system that presents allergic consumers with products that are unsafe for them would do more harm than good.**



**Figure 14 High precision, low recall. The result set contains only safe products. However, some safe products are left out. This is acceptable safety-wise, but imposes unnecessary constraints in the consumer's available choice.**



In both cases, a system's ability to distinguish between relevant and irrelevant, or safe and unsafe for a real-life user, presupposes that the user is able to adequately express his or her need to the system. Some of the examples shown in this thesis illustrate that consumers with

multiple or atypical food allergies or intolerances are unable to convey their needs due to the limited number of allergens available for choice or because the allergens listed are too general to represent the real needs. This may lead to false positive and false negative results, respectively.

For people with food sensitivities, safety is a prerequisite for regarding a product as relevant. Optimal precision is an absolute requirement for an information system aimed at this group. At the same time, the system should provide high recall in order to give every customer access to the largest possible share of the product range.

### **2.2.3 The fuzzy nature of product safety**

For a person with food sensitivities, assessment of product safety is not necessarily a Boolean “yes or no” question. Between the two extremes *definitely unsafe* and *definitely safe*, there may be a wide range of ambiguous cases. Many different factors may impact a person’s assessment of a product, and not all of these factors can be successfully modeled in an information system.

### **2.2.4 Context sensitive relevance perception**

Many different factors affect how individuals perceive the relevance of information material. Some of these factors have already been mentioned. A vast amount of printed and online sources about food allergies and intolerances are available, but a person is not likely to show any interest in these until she finds herself in a problematic situation (Wersig, 1971, as cited by Belkin, 2005) where existing knowledge falls short. A person’s willingness to spend time and effort reading up on the subject will likely depend on whether he or she will have long-term use of the newly acquired knowledge, like Persona A or D, or only needs a short-term “easy fix”, like Persona C.

Even a person who is genuinely interested in acquiring new knowledge will have trouble utilizing many of the available sources in the actual grocery-shopping situation. The information need regarding safety assessment of individual products requires a source that can be utilized with minimal cognitive effort and time consumption. This implies that information needs to be tailored to the specific consumer’s needs in order to limit unnecessary noise.

In the shopping situation, informing a consumer about products that are safe, but not available on the spot may cause frustration. This would typically occur if a consumer has used vendor brochures in order to identify safe products and it turns out that the local shop doesn't carry the items. An information source aimed at helping consumers in the shopping moment should ideally be based on what is physically available at the very time and place he or she is shopping. Products that are excluded from the shop's assortment or temporarily out of stock should be possible to filter out.

### 2.2.5 Data vs. information

In this thesis, I have already used the term "information" many times in different contexts. In everyday language, the meaning of the term is quite fuzzy. Different professional disciplines have traditionally ascribed slightly different meanings to the word. The diversity reflects the particular objectives and viewpoints of practitioners of various disciplines revolving around information, such as journalists, computer scientists, marketers and librarians.

In his PHD thesis, *Information-Kommunikation-Dokumentation* (Eng. trans.: Information-Communication-Documentation), Gernot Wersig identified and classified a wide range of information definitions. A critical analysis of the classes of definitions identified led him to dismiss all but the "Wirkung" definitions. "Wirkung" is a German word that can be translated to "effect" or "impact" in English. The Wirkung definitions are oriented towards the receiver, and emphasize the impact a message has on the receiver in a problematic situation. Wersig proposed his own Wirkung oriented definition: "Information is communication-based reduction of uncertainty." (Wersig, 1971, as cited by Ongstad, 1987)

The personas presented in this thesis illustrate how people affected by food sensitivities experience a problematic situation when they are shopping for food. Increasing amounts of data is made available to consumers through various communication channels. However, only a subset has the potential to influence an individual consumer's decision-making process. In line with Wersig's definition, only data that helps reduce a particular consumer's uncertainty about product safety can be considered information in this context. The same data may have information value for one person, but be perceived as noise for others. This calls for

individualized presentation of available data, bringing attention to that which constitutes information for each individual.

### **2.2.6 Problems with manual assignment of allergens**

Most information services aimed at people with food sensitivities seem to rely on manual assignment of allergens to individual products. Allergens may occur in products either directly as ingredients (e.g. milk or eggs), indirectly as components of other ingredients (e.g. gluten in flour or nuts in chocolate chips), or unintentionally as contaminants from the production environment. In the case of contaminants, the need for manual risk assessment and subsequent assignment of allergen warnings on the products level is obvious.

However, in the case of allergens that occur as a direct consequence of a product's ingredients, relying on manual assignment of allergens to individual products would mean redundant efforts and risk of human error. The same ingredients may occur in multiple products, and relying solely on manual assignment on the product level would mean drawing the same conclusions over and over again.

A computer-based tool could possibly reduce the manual effort involved by identifying known allergenic ingredients. However, the same allergen may occur in a wide range of ingredients, and their names aren't always "telling". A computer-based tool relying on finite lists of ingredients or string matching could thus miss obscure manifestations of allergens.

Humans, on the other hand, may miss allergens as a result of a misdemeanor or insufficient domain knowledge.

### **2.2.7 Lack of standardization regarding practice of allergen warnings**

Many vendors now provide warnings about allergens that may occur as contaminants in the product. Some even provide explicit warnings about allergens that are already reflected in the ingredient list. This way of labeling has the potential to eliminate consumers' need for reading the full ingredient list. However, since the vendor's labeling practice is not explicitly stated, it is hard for the consumer to know whether no warning about an allergen he or she needs to

avoid means that it doesn't occur in the product, or that the vendor presupposes that the consumer reads the whole product declaration, and not just the warnings.

Some vendors voluntarily warn consumers about allergens that are not included in the list of allergens subject to special labeling requirements. Others provide no allergen warnings at all, leaving the consumer wondering whether to interpret this as a sign that the product is free from all allergens or that the vendor does not particularly cater for people with special dietary needs. Some vendors label products as "free from" various allergens, while a great deal of products may be free from the same allergens without explicitly asserting the fact.

The *presentation* of allergen warnings given also varies across vendors. Consumers will often need to look several places on the package to ensure that no relevant information is missed. Some products even carry product declarations in several different languages side by side, so a consumer will have to identify the right one.

### **2.2.8 Vendor-specific sources and systems – lack of industry collaboration**

A wide range of sources and services aimed at people with special dietary needs are available. However, in the shopping situation, having to consult multiple sources and services is obviously very impractical. Brochures and websites tend to be oriented towards specific vendors' product selection and cater for predefined needs that may or may not correspond to individual consumers' actual needs. The fact that different vendors convey the same type of information in different ways means that consumers have to spend extra cognitive effort adjusting to each case. This problem applies to both product declarations and allergen warnings found on physical products' packages and the wide selection of information material and services available. Further standardization and collaboration across the industry is necessary to provide consumers with special dietary needs with a practical aid in the shopping situation.

### **2.2.9 Information loss**

Manufacturers possess unique inside knowledge about their products' recipes, production processes, possible contaminants in the production environment, etc. Individual consumers, on the other hand, possess unique knowledge about the dietary needs they have to take into

account when shopping for food. Unfortunately, manufacturer representatives and consumers are unable to communicate directly during grocery shopping.

Instead, product declarations represent the manufacturer's knowledge about the physical products. The presentation is impacted by current labeling requirements as well as the need to protect trade secrets. Thus, an information loss occurs when the manufacturer's knowledge about a product is translated into a textual product declaration.

When an information system is used to aid the matching of consumers and products, additional sources of information loss occur.

Conveying implicit knowledge about own dietary needs and consequent shopping practices can be difficult even when talking to another human being. After all, many contextual factors may impact the assessment of each individual product in each individual shopping instance. Translating all of this into a set of absolute rules would be extremely difficult because the consumer may not even be conscious of all of the factors that impact her decisions.

When the consumer has to express her needs to a computer-based system, the overly simplified representation constitutes another source of information loss. The interactive information systems discussed in this thesis require users to express their needs as a finite set of allergens. Based on this, the systems operate with a Boolean interpretation of product safety. The user has no way of specifying whether trace amount are OK or not – the systems operate based on their own definitions of what it means that a product contains the given allergens.

In some cases, the allergens available for selection can be too general to fit the users true needs. In other cases, the substances that the user needs to avoid are not even available for choice.

#### *False negative match resulting from information loss (hypothetical example)*

Persona D is shopping on behalf of her lactose intolerant granddaughter. She spots a product she fancies, and wonders whether it's safe for her granddaughter. The product declaration is extensive and in fine print, so she decides to use an information system to aid her decision.

The system doesn't provide *lactose* as an option, so she checks the more general *milk* option. According to the manufacturer's product declaration (which Persona D never reads), the product she considers buying contains *milk protein* (but no lactose). The system warns her that the product contains *milk*, so she puts the product aside and moves on.

This hypothetical example illustrates how information loss may contribute to unnecessary limitations of the number of products that consumers affected by food sensitivities would perceive as safe.

#### *False positive match resulting from information loss (hypothetical example)*

Persona A shops on behalf of her kid who has coeliac disease. She has been advised to stay clear of oats for the first six months. Persona A uses an information system to check whether products are gluten free, because she finds it tiring to read all of the declarations. Whether or not products contain oats is evident from the ingredient lists. Persona A sees a product that her daughter used to like before she was diagnosed, and wonders whether she will still be able to eat it. The product's ingredient list is available both on the physical product and in a digital format that the system has imported. However, because oats are not included in the list of allergens that are subject to special labeling requirements, the system's list of allergens available for choice also omits the substance. The system is thus unable to match products against Persona A's real needs, and she will therefore have to manually double check products that the system presents as safe according to her current settings.

This example illustrates how the value of an information system may be limited if the user is unable to adequately express her needs to the system and the system is unable to utilize the available data in the matching process.

#### **2.2.10 Understanding users' goals**

Not all people are particularly interested in food or nutrition, but they still have to eat. For people with food sensitivities, spending time and cognitive effort reading and interpreting product declarations is merely a means to an end. In the shopping situation, customers need to make informed decisions regarding product safety in limited time. Ideally, the user would prefer a straight yes or no *answer as* to whether a given product is safe, thus eliminating the



need for manually interpreting each product declaration. However, in many cases it would be impossible for a system to predict what the customers own assessment would have been in a given situation. The persona descriptions provide some examples of contextual factors that may affect a user's decision.

Based on the personas representing the diversity of individual needs and the examples of existing systems, it seems impossible to provide each individual user with a clear yes/no answer without sacrificing either precision (presenting products that in reality are unsafe as safe) or recall (presenting products that in reality are safe as unsafe).

### 3 INTRODUCING AN ALTERNATIVE APPROACH – A DECISION SUPPORT SYSTEM BASED ON SEMANTIC WEB TECHNOLOGIES AND LINKED DATA

In the previous chapters, I have demonstrated that consumers with multiple or atypical food sensitivities experience information needs in the shopping situation that existing information systems are unable to accommodate in a satisfactory manner.

In this chapter, I argue that the highly individual and context sensitive nature of this problem calls for a tailored *decision support system* that facilitates users' own assessment of product safety. I proceed to propose a new model based on Semantic Web and Linked Data, which would enable efficient information transfer between vendors and consumers with special dietary needs through such a decision support system.

#### 3.1 Rejecting the Boolean interpretation of product safety

The personas presented in this thesis illustrate the diversity within this customer segment. It seems impossible to provide each individual user with a definite yes/no answer regarding product safety without sacrificing either precision or recall. In my endeavor to develop an alternative solution, I therefore reject the Boolean safe/unsafe approach in favor of a *tailored decision support system*. The system is not meant to *replace* the user's safety assessment, but rather to *support* the manual decision making process.

#### 3.2 Design goals

##### 3.2.1 Reliable precision and recall performance

The system needs to be reliable in its ability to identify risks associated with a given product for a given user, based on the vendors product declaration and the user's individual needs – expressed as a set of allergens. The system should be able to categorize the product as either *unsafe* – if at least one of the allergens *definitely* occurs in the product, *uncertain* – if at least one of the allergens *may* occur and the user may want to assess the risk manually, or *safe* – if no risks have been identified.

Optimal precision, i.e. making sure that all unsafe and uncertain products are identified as such, is an absolute requirement for this system. High recall, i.e. avoiding that products that are actually safe are erroneously categorized as unsafe or uncertain is a secondary, but important goal. It is important to avoid that people who already have a restricted diet incur further unnecessary limitations of their available options.

Both precision and recall performance is essential in order to get users to trust the system.

### **3.2.2 Matching based on semantics rather than text**

Users must be able to adequately express their individual needs to the system and the system must be able to match these needs against product declarations provided by vendors – independent of the actual terms used by both parties. The sought allergen terms rarely occur explicitly in the vendors’ product declarations, but are rather “masked” by a variety of more or less familiar terms identifying ingredients in which they appear. The system thus needs to operate on a semantic level, in order to make the deductions necessary to match the users’ expressed needs against the vendors’ product descriptions.

### **3.2.3 Reducing information overload by differentiating between data and information**

In cases where the system is unable to draw any conclusion about product safety, the system should provide the user with decision support by enhancing relevant pieces of information. In order to avoid “information overload”, only data relevant to the users risk assessment should be emphasized.

### **3.2.4 Integrating data from multiple industry actors**

The system should not be limited to any specific food vendor, as consumers should be able to solve their task without juggling several different information systems. Ideally, the system should also work across different supermarket chains. This implies that the system should be based on Linked Data principles, with shared references to entities in the real world, such as allergens and ingredients.

### 3.3 Decision support systems

#### 3.3.1 Definition

Druzdzel & Flynn define decision support systems (DSS) as “...interactive computer-based systems that aid users in judgment and choice activities.” (Druzdzel & Flynn, 2012, pp. 464).

They also refer to *knowledge-based systems* as a synonym to DSS, emphasizing their “... attempt to formalize domain knowledge so that it is amenable to mechanized reasoning.” (Druzdzel & Flynn, 2012, pp. 462). In line with this, I have chosen to model the domain knowledge using semantic web standards and utilize automatic inference for mechanized reasoning.

Druzdzel & Flynn describe how stress and complexity impacts decision-making. In the shopping-situation, people with food sensitivities have to assess the safety of multiple products under pressure. According to Druzdzel & Flynn “... human intuitive judgment and decision making can be far from optimal, and it deteriorates even further with complexity and stress.” (Druzdzel & Flynn, 2012, pp. 461).

#### 3.3.2 Vast and increasing amounts of data

The amount of data available with the potential to influence decision-making is growing incredible fast. However, the human capacity for processing data and taking things into consideration is limited and stable. In the words of Druzdzel & Flynn “[Decision support systems] are especially valuable in situations in which the amount of available information is prohibitive for the intuition of an unaided human decision maker, and in which precision and optimality are of importance. Decision support systems can aid human cognitive deficiencies by integrating various sources of information, providing intelligent access to relevant knowledge, and aiding the process of structuring decisions. They can also support choice among well-defined alternatives . . .” (Druzdzel & Flynn, 2012, pp. 462)

#### 3.3.3 Identifying parts that can be left to machines

It is useful to analyze which parts of the process can be left to machines and which parts still need to be performed by humans. In the case of food sensitivities, a system can be used to

identify which products are *definitely unsafe* for a given user. By flagging unsafe products as such, the system saves consumers a lot of time reading product declarations in vain.

However, in many cases, the safety of a given product for a given consumer is less certain. It is extremely important to respect the limitations of automated deduction. There will always be some nuances that only humans are able to interpret and assess. Since every consumer has unique needs, there are many cases where it is impossible to predict whether a consumer will regard a product as safe or not.

### 3.3.4 User interface

Druzdzal & Flynn emphasizes the importance of the user interface. “Because DSSs do not replace humans but rather augment their limited capacity to deal with complex problems, their user interfaces are critical. The user interface determines whether a DSS will be used at all, and if so, whether the ultimate quality of decisions will be higher than that of an unaided decision maker.” (Druzdzal & Flynn, 2012, pp. 471).

I have therefore chosen to develop a decision support system that automatically categorizes products as unsafe, uncertain or safe for a given user and emphasizes facts that has *information value* in regard to that user’s manual assessment.

### 3.3.5 Core data set administered by a governmental agency

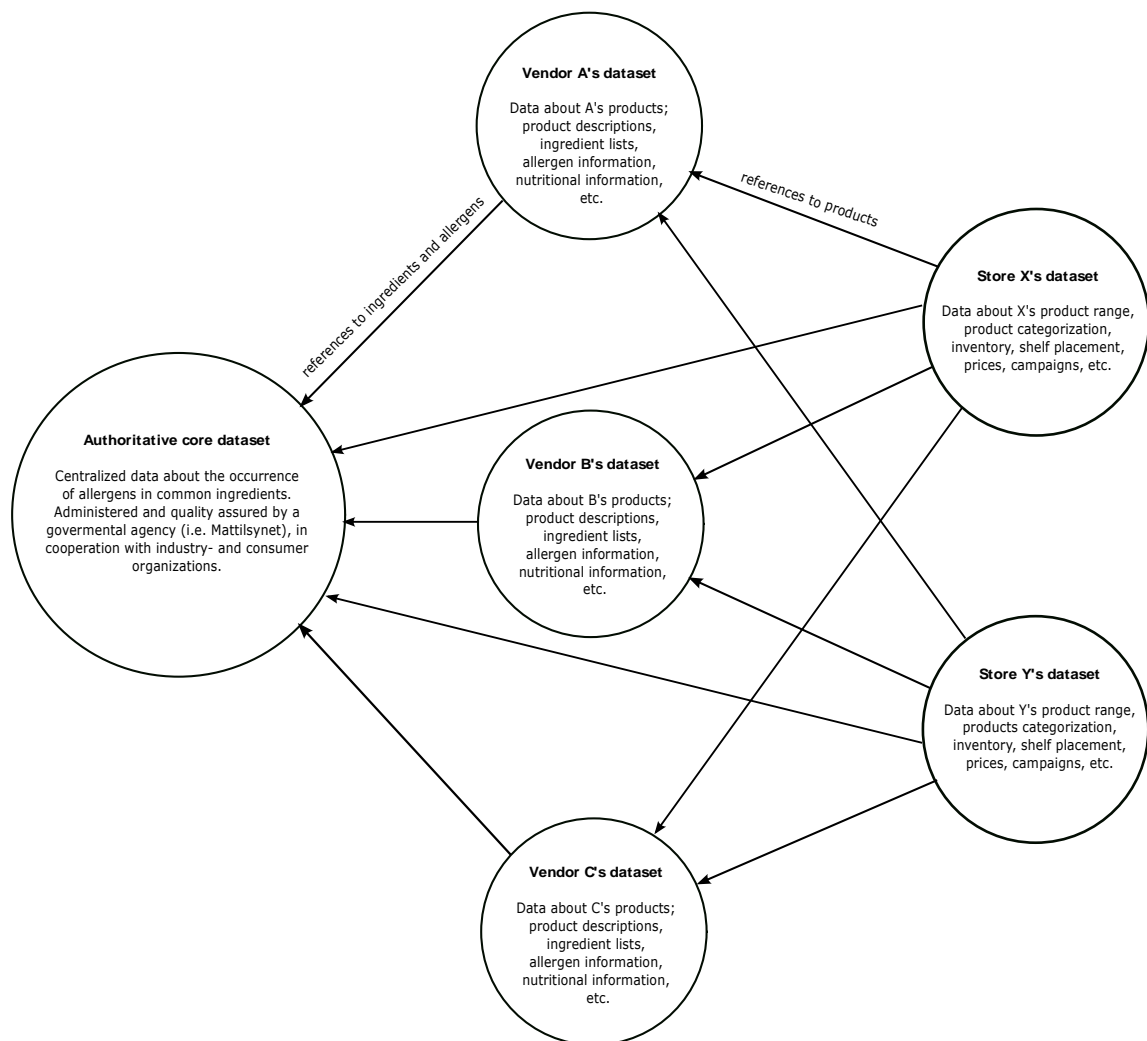
Establishing and maintaining all the data needed to underpin a semantic web based decision support system would be a huge task for any single actor in the food industry. A core data set, holding facts that are common to all, should therefore be administered by governmental agency in collaboration with industry actors. Relationships between ingredients and allergens would be established and maintained by domain experts in one authoritative data core. Vendors would point to these resources in their product descriptions. This approach would reduce the burden on each actor of developing and maintaining data sources expressing detailed domain knowledge. The same data core could underpin both smartphone apps and dynamic websites.

### 3.4 A model based on Semantic Web and Linked Data

#### 3.4.1 Common domain ontology and authoritative knowledge base

My proposed model relies on a common ontology and an authoritative knowledge base holding authenticated data about allergen occurrence in various ingredients.

**Figure 15 A Linked Data based model. Circles represent datasets. Arrows between the circles symbolize that one dataset references RDF resources in another. The largest circle represents the common authoritative data core. Both vendors and stores reference its resources.**



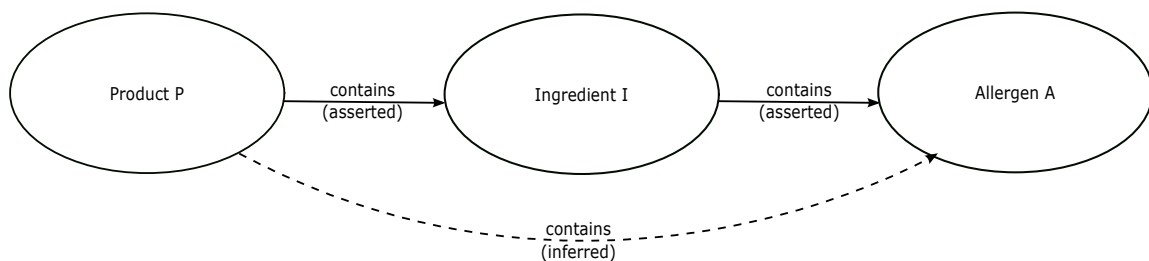
Vendors express their product declarations as RDF instead of free text. The RDF statements refer to ingredient and allergen resources in the authoritative knowledge base and predicates

from the domain ontology. Physical and online grocery stores similarly express their product selections as RDF, referring to vendors' product resources.

### 3.4.2 Automatic inference

The predicates available for expressing relationships between products or ingredients and allergens are defined as *transitive*. This means that allergen occurrence in products can be *inferred* automatically from allergen occurrence in their ingredients.

**Figure 16** The solid arrows represent transitive properties in asserted statements. The dashed arrow illustrates how new statements can be inferred from asserted statements using transitive properties.



### 3.4.3 Preprocessing the data

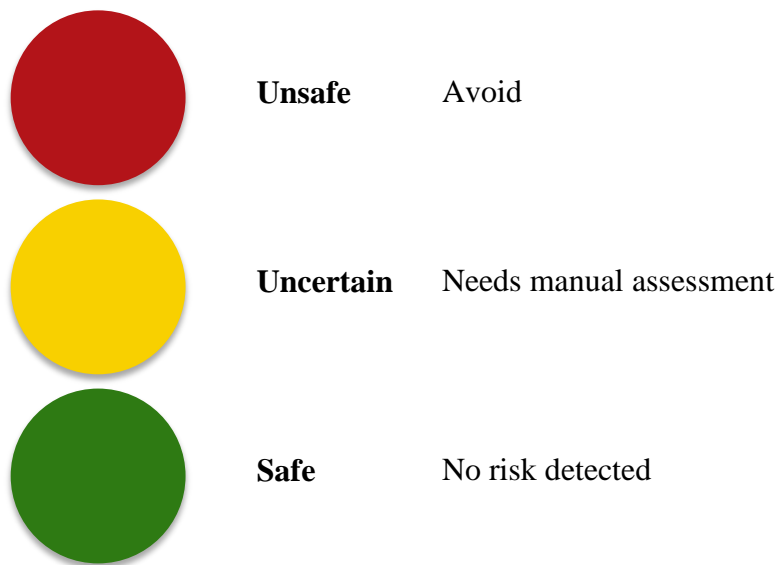
The proposed model underpins a decision support system that preprocesses data in order to efficiently identify and communicate risk associated with products to individual users, based on their dietary needs.

Of course, some products can automatically be “written off” as unsafe without any human intervention. This alone, may save the user a lot of time reading and interpreting products’ declarations; only to find out that one of the last ingredients listed pose a risk. On the other hand, some products have no relation to any of the allergens in question, and are most likely safe. However, many products are uncertain. Based on the user’s expression of his or her own dietary restrictions, products are categorized into *three disjunctive sets*; unsafe, uncertain and safe.

### 3.4.4 Traffic light colors for quick feedback

Traffic light colors are used to quickly communicate whether products are unsafe (red), uncertain (yellow) or safe (green). Users like Persona C would possibly stick to green products and disregard the rest to stay on the safe side, while more experienced and/or dedicated users would at least occasionally want to manually assess the risks associated with yellow products.

Figure 17 Traffic light colors used to convey safety status



If the user opts to view detailed information about a product, tailored decision support is provided by emphasizing data that is likely to affect the particular user's safety assessment, while filtering out noise. That is, the system provides and draws the user's attention to detailed information about any occurrence of the specified allergens, while withholding facts that are irrelevant to the particular user's decision, such as occurrence of other allergens.

### 3.4.5 Smartphone app

The system should be easily available to consumers while shopping. Given the high prevalence of smartphones with built-in cameras and Internet access, utilizing these as a physical medium would be a natural choice. The decision support system is therefore intended to run on the user's smartphone, utilizing the built-in camera to scan product barcodes.



## 4 PROVIDING A PROOF OF CONCEPT

I have developed a prototype application in order to provide proof of concept for my proposed model. For the purpose of this thesis I have chosen to implement the prototype as a *web application*, rather than a *smartphone app*. The prototype solves the core problem of matching users' needs – expressed as a set of allergens they need to avoid – with vendors' product descriptions. The application classifies products as *unsafe*, *uncertain* or *safe*, and quickly communicates the status of each product using familiar traffic light colors – *red*, *yellow* and *green* – using CSS formatting. Tailored decision support is provided on demand, emphasizing facts that are likely to affect the users assessment of uncertain products.

At this point, product descriptions are not available as Linked Data. I have therefore put together a small dataset representing some products available on the Norwegian market for the purpose of the prototype.

### 4.1 Knowledge prerequisites and references

The following presentation assumes basic knowledge of the ideas behind Semantic Web and Linked Data, and familiarity with RDF and SPARQL. Readers new to this field are referred to a popularized introduction published in Scientific American (Berners-Lee, Hendler & Lassia, 2001), as well as Berners-Lee's guidelines for publishing Linked Data (Berners-Lee, 2006). The World Wide Web Consortium (W3C) provides comprehensible introductions to the Resource Description Framework (RDF) and SPARQL query language (W3C, 2004c, 2008) and related W3C recommendations.

My work with ontology development and Semantic Web programming is based on insights gained from textbooks such as *Semantic Web for the Working Ontologist* by Allemang & Hendler (2011), *Semantic web programming* by Hebler, Fisher, Blace & Perez-Lopez (2009), *Programming the Semantic Web* by Segaran, Evans & Taylor (2009) and *Web Technologies: A Computer Science Perspective* by Jackson (2007) – as well as documentation published by W3C (W3C, 2004a, 2004b, 2004c, 2006, 2008, 2009a, 2009b) and the Jena community (Apache Jena, n.d.; McBride, 2010).

## 4.2 Prototype development process

In my attempt to provide proof of concept for the proposed model, I decided to work with ontology development and application layer development in parallel. This iterative process made me able to recognize undesired consequences of my design choices and make the necessary adjustments along the way.

I quickly realized that *simplicity* was a key success factor, because overly complex SPARQL queries in the application layer would make it near impossible to heuristically verify that the application would provide the expected result in all possible cases. On the other hand, an overly simplified ontology would make it impossible for data providers to describe their products without losing nuances that might be relevant to the end user.

OWL modeling involves use of *sub-classes* and *sub-properties*, which means that resources and the relationships between them can be expressed with a high degree of specificity and still “respond” to more general queries, characterizing resources and relationships by their super-classes and super-properties.

In this thesis, I have developed an ontology that satisfies both the need for simplicity and the need for *minimizing information loss* between the information providers and the end users. The ontology could easily be extended with further use of sub-classes and sub-properties, or with additional classes and properties, describing other aspects of the domain that are outside the scope of my prototype.

## 5 ONTOLOGY DEVELOPMENT

### 5.1 The Web Ontology Language (OWL)

The Web Ontology Language is a W3C recommendation for defining ontologies that underpins the Semantic Web (W3C, 2004a). An OWL ontology describes a domain in terms of *classes*, *properties* and their *instances* in a way that machines are able to interpret. The formal semantics of OWL defines how logical consequences can be derived from a given ontology, entailing facts not literally stated. The process of deriving such entailments is called *inference*, and can be implemented in different ways. OWL provides mechanisms for combining multiple distributed ontologies. (W3C, 2004a, 2004b)

Different sub-languages of OWL are available, each with their own advantages and drawbacks. For this project I have chosen to use the *OWL DL* (*DL* is an acronym for *Description Logic*), because it provides a high degree of expressiveness, but with some constraints that guarantee that inference computation will finish within *finite time* and that *all entailments* will be computed. (W3C, 2004b)

### 5.2 Protégé ontology editor

I have used Protégé, which is a free open source ontology editor, to develop and maintain the ontology for this project (Protégé, n.d.). After trying out different versions, I found that Protégé version 3.4.8 had most of the features I needed, such as a comprehensible GUI with adjustable forms for entering instance data, and support for transitive properties and ordered lists.

### 5.3 Data modeling

Data modeling involves making a representation of reality and always involves some degree of simplification. It is therefore important to evaluate which aspects of the domain need to be expressed in the ontology.

### 5.3.1 Classes

#### 5.3.1.1 *Product, Substance, Ingredient and Allergen*

In this setting, everything revolves around *products* and the occurrence of *allergens*. Thus *products*, *ingredients* and *allergens* stand out as real-world entities that have a natural place in the ontology. Since some resources (e.g. eggs or nuts), may occur both as ingredients or as allergens, I have chosen to generalize the two classes `Allergen` and `Ingredient` into a common super-class named `Substance`, which again sub-classes the external class `skos:Concept`.

#### 5.3.1.2 *Ingredient list*

The food labeling regulations are developed to accommodate consumers' need for information, without forcing food manufacturers to reveal all of their trade secrets. Food manufacturers are thus not required to disclose their complete recipes, but instead required to specify each product's ingredients *in descending order* (Merkeforskriften, 1993). For a consumer with food sensitivities, the *amount* of a substance that a product contains may be decisive to whether or not she will risk eating the given product. As an example, a coeliac sensitive to wheat starch may choose to avoid a gluten-free loaf based mainly on wheat starch, but still choose to eat a pate that contains a small amount of the same substance. Both the *concentration* of the substance in the product and the *amount* of product consumed will affect the user's decision.

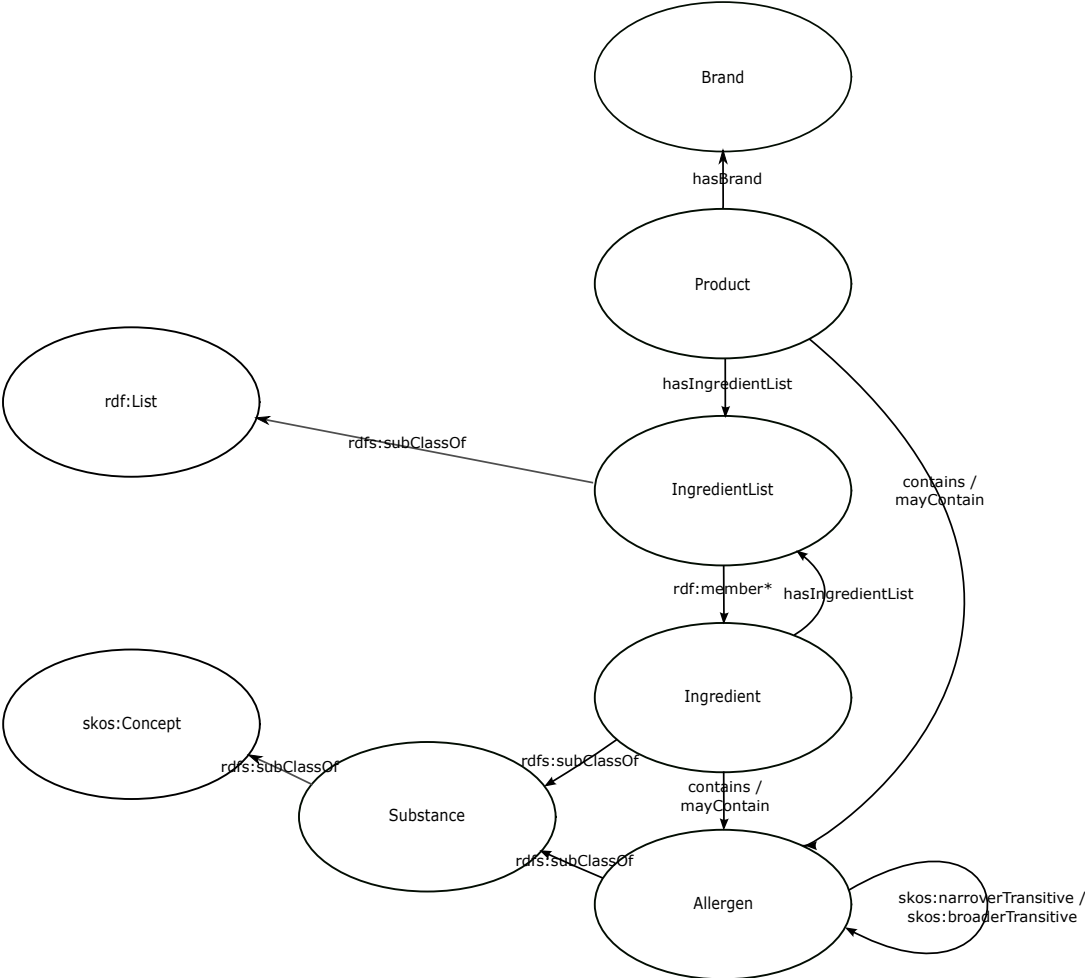
Since the vendor is normally not required to specify the exact *amount* of each ingredient, the *order* of the ingredients specified in a product declaration is the best available indicator of the relative amount of a given substance in a product. The order of the ingredients is thus clearly of high relevance to a consumer assessing a product, and needs to be conserved in the data model for presentation to the user as an instance of the class `IngredientList`.

An OWL-based data model is a graph, where "nodes" (RDF-resources and literals) are connected by "edges" (predicates) in a multi-dimensional space. Such a graph has no inherent order when mapped to a one-dimensional representation, such as an ordered list. This can be observed when Protégé or Jena serializes a graph, as the triples may show up in completely different orders from one time to the next.

The RDF-vocabulary supports a class named `rdf:List`, which can be used to store ordered lists in RDF. An `rdf:List` always consists of two elements; a `rdf:first` element and a `rdf:rest` element, which in turn refers to another `rdf:List`-element (W3C, 2004c). The `IngredientList` sub-classes `rdf:List`, and thus inherits its structure and properties.

I found the current user interface for entering lists in Protégé to be quite impractical for products with more than a couple of ingredients, since each entry in itself comprises a new list and thus requires a new window. Being an open source tool, Protégé could be developed further in the future to provide a more practical user interface for this task, hiding unnecessary implementation details from the user.

Figure 18 Ontology outline



### 5.3.1.3 Users

I have chosen not to represent *users* in the ontology at this point, because the current prototype application does not store user data in between sessions. Before extending the ontology to represent user data, it should first be investigated whether the RDF format is suitable for storing sensitive data.

### 5.3.2 Properties

Since the model is meant for decision support, the *nature of the relationship* between a product and an allergen is also important to communicate. It is not sufficient to tell the user that Product P has a relationship to Allergen A, because the user needs more detailed information to assess the safety of the product.

It is important that the properties in the RDF model are specific enough to express all vital information, in order to avoid information loss. Any nuance relevant to an individual's evaluation of a product needs to be communicated so that the user can make an informed decision. For instance, a user may decide that a product *produced alongside* an allergen may be worth the risk, but choose not to eat a product with ingredients *derived from* the same allergen. After all, each individual should be able to make their own decision, based on experience of how sensitive they are, and to what extent they are willing to take risk in a given situation.

#### 5.3.2.1 Certain and uncertain relationships

I have chosen to establish two main properties between products or substances and allergens; `contains` and `mayContain`. These two properties are both *transitive*, and are used as super-properties for more expressive sub-properties. The `contains`-property expresses a *certain relationship* between a product (or substance) and an allergen (or other substance), and will deem the product *unsafe* to a user who is sensitive to the given allergen. The `mayContain`-property, however, expresses an *uncertain relationship* between a product (or substance) and an allergen (or other substance). That is, a *risk factor* that may or may not lead an allergic person to avoid the product. Any sub-property of `mayContain` will lead the model to

categorize a product as *uncertain* for a person allergic to the allergen in question. It will then be up to the user to assess the risk and whether she is willing to take it or not.

Both the `contains`-property and the `mayContain`-property and thus all of their sub-properties can be assigned either directly to a `Product`-resource or to a `Substance`-resource (or both). Typically, the vendor will specify an ingredient list where one or more `Ingredient`-resources have a relationship to some of the allergens in question. Based on the ontology and the available data, the model will *infer* that the product has a relationship to these allergens. However, the vendor may identify risk factors that cannot be derived from ingredients, such as *contaminants in the production environment*. Therefore, the vendor can optionally assign allergen relationships directly to the product level.

5.3.2.2 *Transitive properties*

My model relies on the use of transitive properties. Figure 19 illustrates how new statements can be automatically inferred from asserted statements based on transitive properties. Figure 20 shows the transitive properties that occur in the ontology and how they relate to each other.

**Figure 19** The solid arrows represent asserted statements. The dashed arrow illustrates how new statements are automatically inferred based on transitive properties.

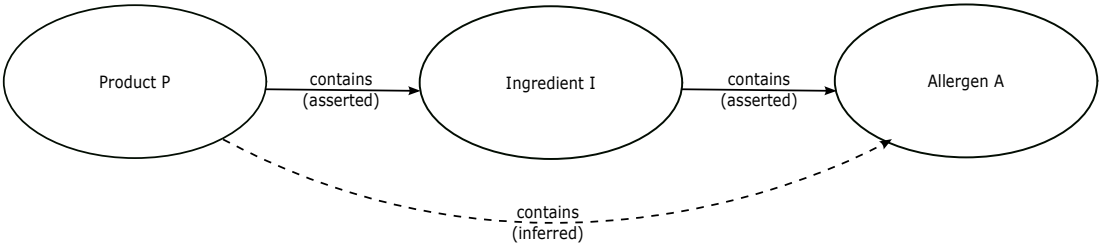
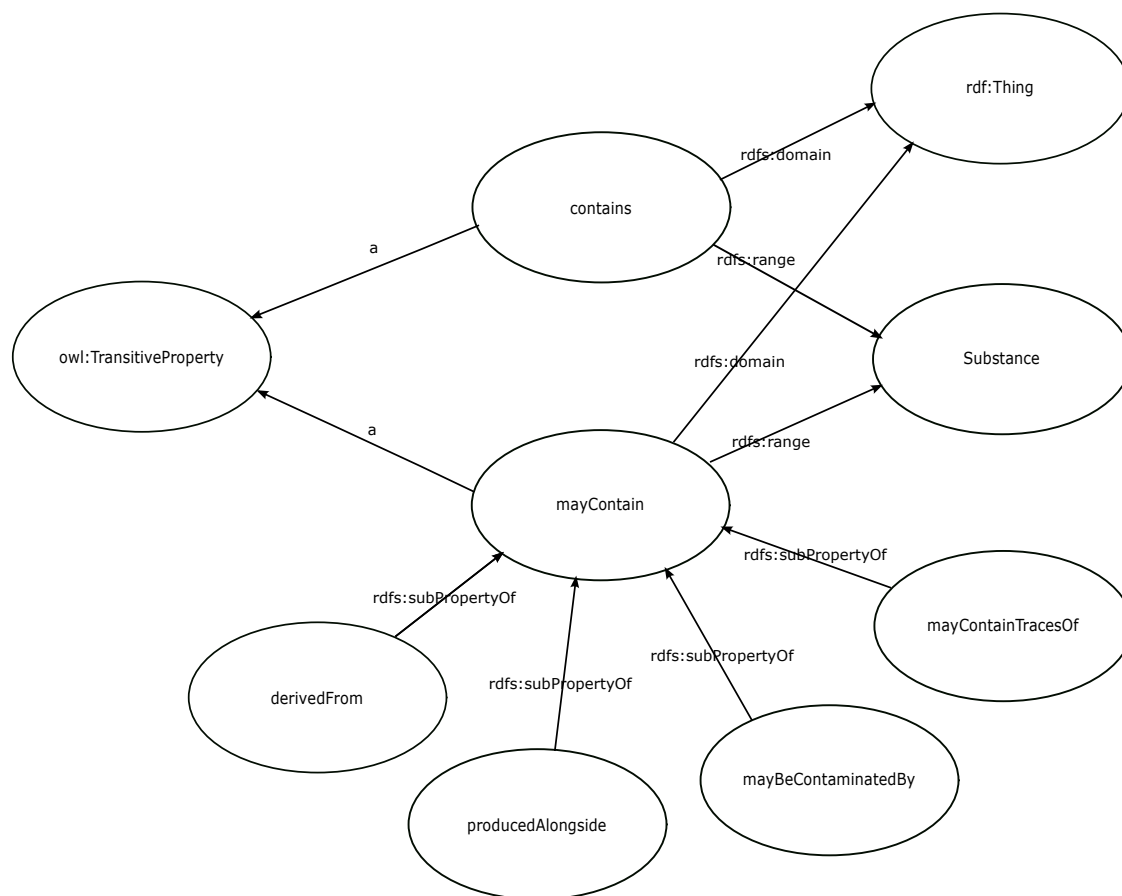


Figure 20 Transitive properties used in the ontology.



Other properties used can be seen in the ontology outline in Figure 18.

### 5.3.3 Semantic relationships between allergen instances

I have chosen to express hierarchical relationships between some allergens. The labeling requirements oblige vendors to label occurrence of milk and nuts. However, both user needs and product declarations tend to be more specific (e.g. milk protein or hazelnut). Utilizing the specificity of the available product declarations and enabling users to express their actual needs with higher granularity (i.e. lactose instead of milk) is an important measure to avoid unnecessary limitation of users' selection.

The Substance-class sub-classes the skos:Concept-class. I did this because the SKOS-ontology (W3C, 2009a, 2009b) provides a wide set of properties that can be used to describe semantic relationships between resources. I decided to use the properties



`skos:broaderTransitive` and `skos:narrowerTransitive` to describe the relationship between nuts (in general) and all the specific types of nuts, such as cashew nuts, hazelnuts etc. and between milk (in general) and the two components that people tend to be sensitive towards; lactose and milk protein.

A person allergic to all nuts will thus not need to specify every single type to make sure that all are eliminated. On the other hand, a person who is only allergic to a specific type of nut will still be presented with products containing other types of nuts. The vendors, on the other hand, can assign relationships to either the general or specific allergens, depending on what is appropriate in each case.

All SPARQL-queries in the application layer take into account that a user should be warned about occurrence of any of the specified allergens' broader or narrower allergens. This provides additional safety, compared to skim-reading declarations manually and possibly missing unexpected phrasings representing an allergen-variant.

The hierarchical relationships between allergens facilitate a user-friendly presentation layer, where hierarchically related allergens are presented together.

## 5.4 Problems and workarounds

As mentioned before, I have aimed to limit the complexity of the SPARQL queries in the application layer, in order to make troubleshooting and verification manageable. I therefore implemented a couple of “workarounds” in the application layer that algorithmically construct and add additional RDF-statements to the dataset before requests from users are handled. The additional statements ensure that the subsequent SPARQL queries can rely on inference over transitive properties and that the query formulation doesn't have to account for all specialized scenarios.

### 5.4.1 Remodeling the relationship between products and ingredients to enable inference

This nested list structure of the ingredient list conserves the order of elements, but is regrettably ill combined with automatic reasoning.

Fortunately, I was able to solve this by making a workaround in the application layer. The transitive properties are essential in order to compute whether a given product has any relationship to a given allergen. However, when the product's ingredients are stored as `rdf:List`, the *direct relationships* between the product and its individual ingredients are lost. In the application layer, I solve this by initially constructing a small graph that re-establishes the direct connection between each product and the members of its ingredient list (i.e. the ingredients) as `contains`-relations.

```
CONSTRUCT {
    ?product ao:contains ?ingredient .
}
WHERE {
    ?product a ao:Product .
    ?product ao:hasIngredientList ?ingredientList .
    ?ingredientList list:member ?ingredient .
}
```

The resulting graph is added to the application's ontology model, so that the transitive properties can be exploited in subsequent SPARQL queries. Similarly, statements asserting that *each substance contains itself* are added to the model. These statements make it unnecessary to take into account when formulating subsequent SPARQL queries that some substances, such as eggs and fish, can occur either as ingredients or as allergens. This is yet another workaround since I was unable to define *contains* as a *reflexive property* in Protégé.

Since SPARQL queries do not differentiate between statements originating from the data source and statements added later on, the application layer actually holds two different ontology models. The original model is based on the serialized dataset for Protégé, and is queried whenever exclusively *asserted statements* are of interest. This is currently utilized to render allergen information according to the vendor's explicit warnings. The thought behind this is that the user *may* assign different weights to warnings made explicit by the vendor as opposed to warnings based on inference over the transitive properties in the dataset. After all, some broadly defined substances may produce warnings because contamination of a given allergen cannot be ruled out, and probably pose a minor risk for most people.

In other cases, the extended ontology model that is based on the original model as well as inferred statements based on the transitive properties from the ontology and the algorithmically added statements mentioned above, is queried.

## 5.5 Discussion and suggestions for future work

### 5.5.1 Issues with “free from” modeling

Some vendors explicitly label their products as e.g. “free from gluten” or “free from milk”. I have not provided a corresponding “free from” property in the ontology at this point.

Theoretically, products asserted to be “free from Allergen A” should constitute the logical complement of products that “contains Allergen A” or “may contain Allergen A”. Maintaining this logical divide in the dataset would require non-trivial, rule-based reasoning.

I could have added a “is free from” property to the ontology straightforward, but then nothing would stop vendors from stating that “Product P is free from Allergen A” while inference would uncover that “Product P contains Allergen A” or “Product P may contain Allergen A”. Such conflicts would typically occur if the vendor has added ingredient resources to the ingredient list that has a relationship to the given allergen. I have encountered several examples of such conflicts in text-based product declarations, i.e. products that are labeled “gluten free” while the ingredient list shows that the product contains “wheat flour”. It is impossible for consumers to know whether the “free from” claim or the ingredient list is correct. A Semantic Web based system should, however, be able to detect and eliminate such logical inconsistencies. One solution would be to automatically detect conflicts as products are being entered into the dataset, stopping vendor representatives from entering contradictory data.

Products that are labeled “free from” tend to be developed with special care for people with food sensitivities. Automatically adding statements asserting that all products that neither “contain” nor “may contain” a given allergen (according to the inference model) are “free from” the given allergen would be misleading. Particularly sensitive individuals would probably prefer products that are both classified as “safe” by the decision support system (because no risks were detected) *and* explicitly labeled “free from” the allergens in question.

### 5.5.2 Consequences of semantic relationships between allergen instances

In spite of the mentioned workarounds, the SPARQL queries in the application layer are still quite extensive. The main factor complicating the query formulations is that they account for the fact that a product or ingredient may have a relationship to a broader or narrower form of a checked allergen. This too could possibly be simplified by adding extra statements to the model. However, this would involve some degree of *interpretation*: If the vendor states that a product contains nuts (general allergen), this may or may not mean that the product contains all of the narrower types of nuts. Adding statements claiming so could thus mislead the user. If the vendor has referred to a general allergen, the correct interpretation is that any of its more specific forms cannot be ruled out.

However, I have chosen to simplify the *presentation* of cases like these so that the user is always presented with the allergen term that she checked (e.g. nuts), rather than the form that actually occurs in the data set (e.g. hazelnuts). The idea behind this is that reusing the checked term makes it easier for the user to spot it when skim reading the page. Since the semantics of the data core is left intact, the presentation of the data could easily be changed.

Another problem with adding extra statements to the data set is that they could easily lead to erroneous deductions either by automated inference or in single queries. If occurrence of lactose (specific allergen) implies occurrence of milk (general allergen) and occurrence of milk implies occurrence of milk protein (specific allergen), we would get the conclusion that occurrence of lactose implies occurrence of milk protein, which is obviously wrong.

### 5.5.3 Semantic relationships between ingredient instances

Originally, I wanted to express semantic relationships between related ingredients, as well as related allergens, using the SKOS vocabulary. My hope was that this would enable more efficient assignment of allergens to ingredients by exploiting hierarchical relationships and inference based on transitive properties. By explicitly asserting a relationship between a general ingredient and an allergen, the same relationship could automatically be inferred for the more specific ingredients, thus saving valuable resources.

However, I soon realized that this could lead to unexpected errors, due to processes that reduce or completely eliminate the presence of components normally found in a substance. For instance, while cow's milk naturally contains lactose, milk products like lactose free milk and some cheeses do not. Likewise, most ingredients based on wheat contain gluten, but wheat starch and substances such as glucose and dextrose are considered gluten free according to current regulations. That is, the gluten content is within the allowed concentration. Conversely, even though most gluten-containing products contain wheat, some contain gluten from other sources such as barley or rye, and may thus be safe for a person who is *allergic* to wheat but not a coeliac. Examples like these lead me to believe that assignment of allergens to hierarchically related groups of ingredients could easily contribute to unnecessary limitation of users' options. This is contrary to what I am trying to achieve.

Even though I decided not to model semantic relationships between ingredients at this point, it should be mentioned that this approach would have considerable advantages for the user interface that needs to be developed for the vendor representatives. The semantic relationships between ingredients could support browsing and presentation of search results, helping these users identify the correct RDF resources in cases where similar substance resources may easily be mistaken for one another.

Even though manual assignment of allergens to each individual ingredient is quite resource intensive, I still think this practice would be more efficient and possibly safer than relying solely on manual assignment of allergens to individual products. Clearly, both methods are vulnerable to human error. However, when allergens are assigned to ingredients once and for all, an error will affect all products containing that ingredient, and thus have greater consequences than an error regarding a single product only. On the other hand, this may increase the likelihood that the error is identified and corrected within reasonable time, given that a good feedback system is provided.

My ontology only includes the classes and properties that were absolutely necessary to solve the core problem I aimed to solve in my thesis.

#### 5.5.4 Refining the ontology to take into account that products may change over time

As the developers of the CheckContent application have pointed out, a product may undergo changes in either recipe or production environment without being assigned a new barcode. (CheckContent, n.d.). The product name, brand, description and packaging may remain the same, thus making it difficult for a customer to detect any change.

In a real world setting, it would therefore be necessary to distinguish between a branded product and different versions of it. This refinement of the ontology could be inspired by the FRBR model developed within the library sector to express the relationship between a work and its expressions, manifestations and items. (IFLA Study Group on the Functional Requirements for Bibliographic Records, 2009). Careful analysis would determine what classes and properties are needed and what “levels” in the model the different properties (GTIN code, name, brand, ingredients, allergens etc.) should be assigned to.

The product version problem can't be solved adequately until regulators require vendors to assign a new GTIN code whenever changes in the recipe or production circumstances occur. Until that time, my model could be adapted to give the user a warning whenever different versions of a product exist. The user could be presented with an easy way of determining the version of the product she is holding, such as batch number or production date intervals, and be guided to the correct instance based on that.

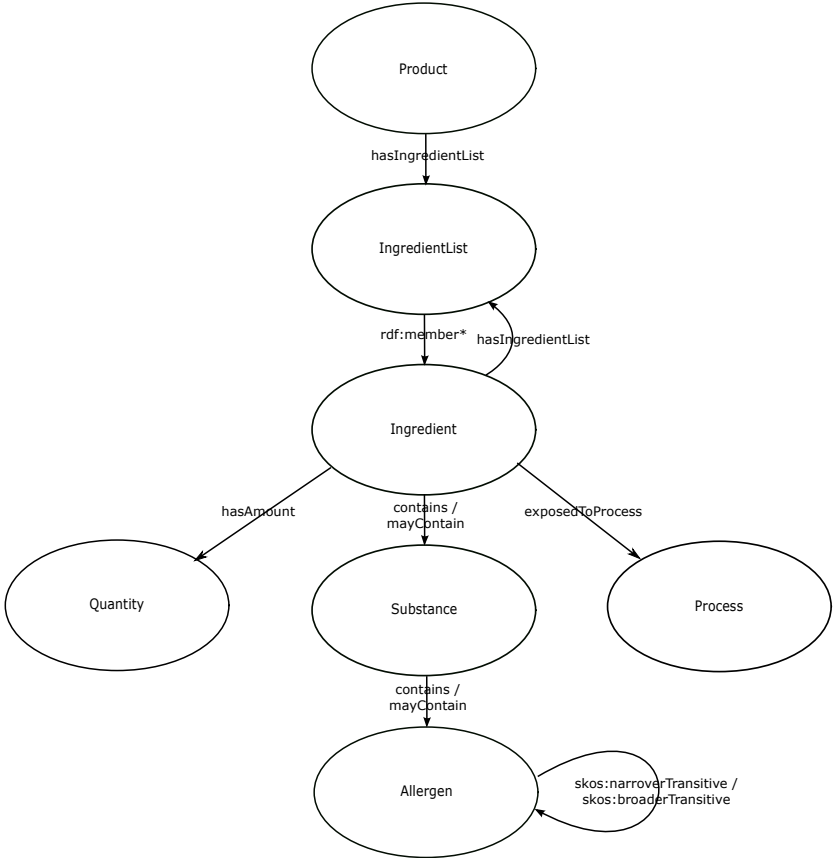
Similar refinement to the ontology would be necessary to support data about food traceability. Whereas I have only included one class to represent the product brand, it would be necessary to distinguish between different actors (manufacturer, distributor) and physical production environments etc. This is a complex modeling task that goes beyond the scope of this thesis.

#### 5.5.5 Modeling of binary vs. N-ary relations

Food declarations are not just aimed at people with food sensitivities. They play an important role in consumers' assessment of food *quality*. In some cases, vendors are obliged to specify the relative *amount* of key ingredients in a product. Examples are the amount of chicken meat in a chicken salad or the amount of blueberry in blueberry yoghurt.

In OWL, properties link together two individuals (or an individual and a value). Properties are thus said to express *binary relations*. This is sufficient for expressing simple facts like “Product P contains Substance S”. However, in some cases it may be desirable to model more complex relations like “Product P contains 5 grams of Substance S”. This is an example of a so-called *N-ary relation*. W3C has provided patterns for representing N-ary relations in OWL (W3C, 2006). In the case of my ontology, it would require remodeling the relation between a product and a substance by remodeling the class `Ingredient` so that it would have a relation to a substance and a relation to a specification of the amount.

**Figure 21** A possible re-modeling to enable N-ary relations. Observe that model allows expression of several aspects of each ingredient, not just its composing substance.



```
Product P contains Ingredient I 7  
Ingredient I contains Substance S  
Ingredient I hasAmount "5 grams" 8
```

This type of data structure would also enable expression of other aspects of an ingredient, such as the fact that an ingredient is *organic* or that it has been processed in a certain way (*smoked*, *irradiated* etc.). By reusing the contains-property, I would still be able to exploit the transitivity to infer that Product P contains Ingredient I. However, I have not remodeled my ontology to support N-ary relations at this point, partly because labeling exact amounts is rarely required and partly because it would make the inference and SPARQL-queries slightly more complicated to verify.

#### 5.5.6 Reuse and mapping to existing vocabularies and ontologies

Reuse of established vocabularies and ontologies is strongly encouraged in Semantic Web development because it makes it easier to link together distributed data sets (Heath, T., & Bizer, C., 2011). Existing vocabularies and ontologies can be used directly, combined with each other, extended with additional classes and properties or mapped to more specialized vocabularies or ontologies. There are many considerations to make in this process, which is why I decided to first model the core ontology for this project “from scratch” using pen and paper, and only then proceed to consider which existing ontologies and vocabularies could provide some of the desired features.

A couple of well-adopted vocabularies stand out as good candidates for future integration with the core ontology:

*Good Relations* (GR) is a controlled vocabulary for providing detailed descriptions of *product offerings* on the web (Hepp, 2011). Google, Yahoo! and other major players currently utilize GR-encoded data embedded in web pages to provide users with more precise search results than unstructured free text pages would allow. The product and brand classes in my ontology

---

<sup>7</sup> In this simplified example, I have abstracted away from the fact that in my ontology, products and ingredients are not directly connected, rather indirectly via the `ingredientList`.

<sup>8</sup> The literal in the last statement could be replaced by a resource in order to preserve the semantics unit of measurement and number value.



could easily be mapped to or replaced with corresponding classes in GR. GR also provides classes and properties for expressing facets that I have omitted from my core ontology, such as GTIN code, price, quantity, dimensions, reviews and information about business entities.

*Quantities/Units/Dimensions/Types* (QUDT) is an alternative vocabulary that could be used to express quantitative attributes of products, such as weight (Allemang & Hendler, 2011).

*The Simple Knowledge Organization System* (SKOS) is a W3C Recommendation for representing knowledge organization systems such as controlled vocabularies, thesauri and taxonomies. (Allemang & Hendler, 2011). In my ontology, SKOS is used to express thesaurus relations between `Allergen` instances. Similarly, SKOS properties such as `skos:related`, `skos:broader` and `skos:narrower` could be utilized to improve browsing and searching of `Ingredient` resources in the authoritative data core. SKOS could also be used to solve issues relating to *synonymy*, by defining *preferred*, *alternate* and *hidden labels* for the same substances.

The RDF language itself provides a simple way of specifying the *language* used in literal values, so the same ontology and instance data could be used across multiple countries.

## 6 APPLICATION LAYER DOCUMENTATION

### 6.1 Implemented use cases

The application prototype currently implements two use cases that represent the core capabilities of the suggested model. Both use cases start with the same steps:

1. A user enters the web site.
2. The system presents the user with a list of allergens and prompts the user to check the ones that apply to her.

Figure 22 Create profile – Persona A

Welcome

**Create your profile**

First name:

Last name:

E-mail address:

**Check everything you want to be warned about**

- Celery
- Citrus
- Crustaceans
- Eggs
- Fish
- Gluten
- Legume (all)
- Beans
- Lentils
- Peanuts
- Peas
- Lupine
- Milk (all)
- Lactose
- Milk protein
- Molluscs
- MSG
- Mustard
- Nuts (all)
- Almond
- Hazelnuts
- Peanuts
- Walnuts
- Oats
- Paprika
- Sesame
- Shellfish
- Soy
- Sulfite
- Tomato
- Wheat

Figure 23 Create profile – Persona B

Welcome

**Create your profile**

First name:

Last name:

E-mail address:

**Check everything you want to be warned about**

- Celery
- Citrus
- Crustaceans
- Eggs
- Fish
- Gluten
- Legume (all)
- Beans
- Lentils
- Peanuts
- Peas
- Lupine
- Milk (all)
- Lactose
- Milk protein
- Molluscs
- MSG
- Mustard
- Nuts (all)
- Almond
- Hazelnuts
- Peanuts
- Walnuts
- Oats
- Paprika
- Sesame
- Shellfish
- Soy
- Sulfite
- Tomato
- Wheat

3. The user checks one or more allergens and submits the form.
4. The system presents the user with her current settings and available actions; scan an individual product or categorize all products.

Figure 24 View profile – Persona A

Display filter

**Your profile**

Name: Persona A  
E-mail:  
Filter:  
• Gluten  
• Lactose  
• Oats

**NB! Not all vendors label trace amounts of allergens. If you are highly sensitive, be cautious!**

Edit profile

**Find out what products are safe for you**

Scan an individual product:  Go!

or  
Categorize all available products

Figure 25 View profile – Persona B

Display filter

**Your profile**

Name: Persona B  
E-mail:  
Filter:  
• Gluten  
• Wheat

**NB! Not all vendors label trace amounts of allergens. If you are highly sensitive, be cautious!**

Edit profile

**Find out what products are safe for you**

Scan an individual product:  Go!

or  
Categorize all available products

### 6.1.1 Scan an individual product

If the user chooses to scan a product, the interaction continues as follows:

- 5a. The user scans a product (barcode or QR-code) and presses “Go!”.
- 6a. The system presents the user with detailed information about the given product, such as indication of product safety (green, yellow or red) and an ingredient list where any occurrence of the allergens in question is pointed out. The nature of the relationship between the product or individual ingredients and the allergens is stated and enhanced by CSS formatting.

Figure 26 Product details – Persona A

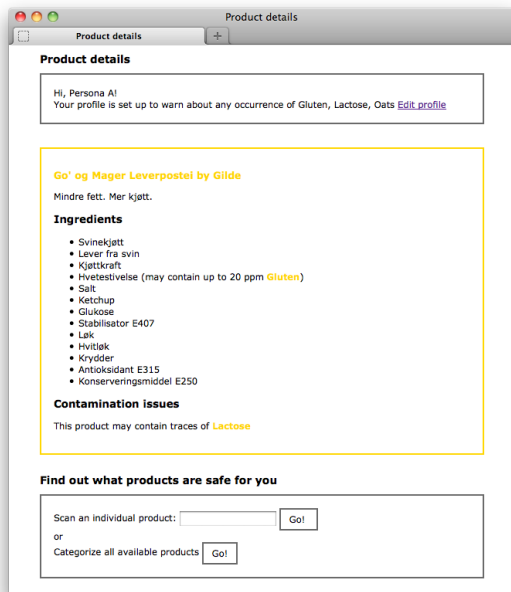
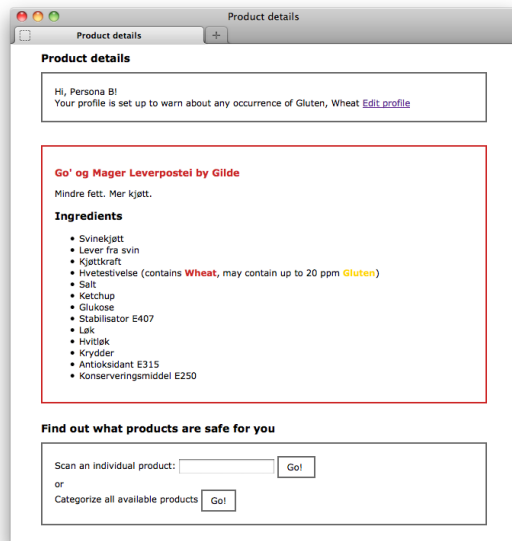


Figure 27 Product details – Persona B



### 6.1.2 Categorize all products by safety

If the user chooses to categorize all products, the interaction instead continues as follows:

5b. The user presses the “Go!”-button following the text “Categorize all products”.

6b. The system presents the user with three lists of products. CSS-formatting is used to communicate which products are safe, uncertain and unsafe respectively.

Figure 28 Categorized products – Persona A

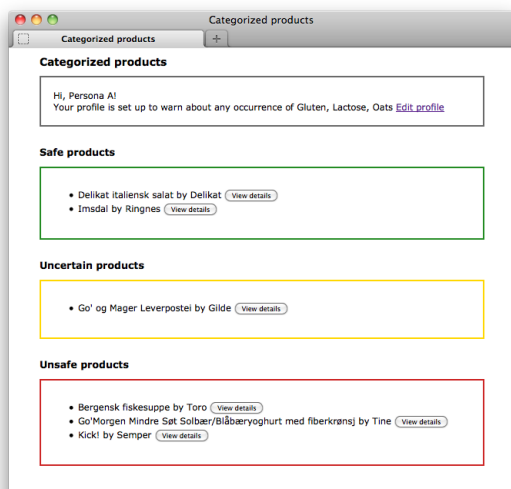
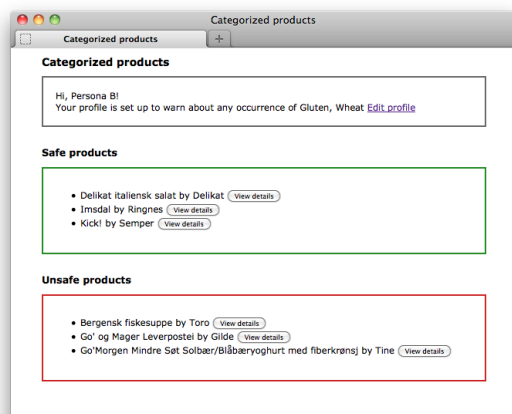


Figure 29 Categorized products – Persona B



7b. The user may proceed to request detailed information about a product, in which case the system presents her with the same page as if she has scanned that product (see previous use case).

## 6.2 Technology choice

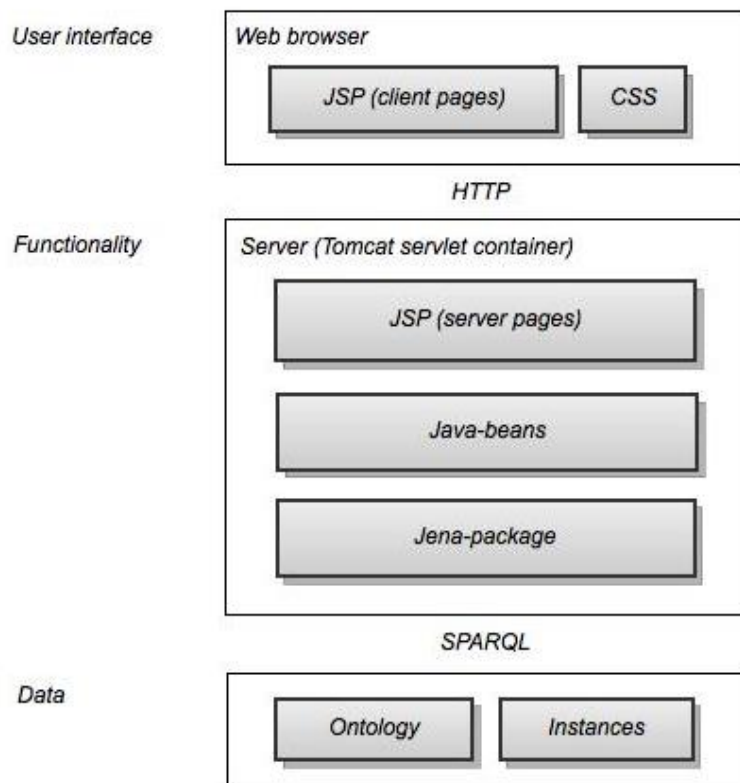
I have chosen to implement the prototype as a web application based on Java and JSP, running in an Apache Tomcat environment. The main rationale for this decision is that Java provides extensive support for working with RDF through the Jena package. (Jena: A Semantic Web Framework for Java, n.d.). By implementing Java classes according to the JavaBeans standard, I enable users to interact with the application via dynamic JSP pages. (Jackson, 2007). The client side is HTML/CSS, which is device independent and easily portable to mobile devices.

## 6.3 Overall application architecture

The overall architecture of the prototype, where the user interface, functionality and data are separated into independent layers, is illustrated in Figure 30. Low coupling between the layers is a design goal that makes it possible to make changes in one layer without redesigning the others. (Mathiassen, Munk-Madsen, Nielsen, & Stage, 2000, pp. 272-273.).

The user's web browser communicates with the web server over the HTTP protocol. The application running inside the web server queries the RDF data using SPARQL.

**Figure 30 Overall application architecture**



## 6.4 JSP and Java beans

The source code of the JSP pages contains a mixture of HTML-tags and specialized JSP-tags. The JSP-tags are resolved on the server-side before the resulting web page is served to the client. Each JSP page initially declares which JavaBeans are in use, which Java class each bean instantiate and its scope (i.e. application, session or request). The scope determines both the accessibility and lifetime of a bean.

The example below shows the declaration of a bean identified as `user`, which instantiates the `User` class. The user bean will have the lifetime and accessibility of an individual HTTP-session.

### Example JSP source code (excerpt)

```
<jsp:useBean id="user" class="myBeans.User" scope="session" />
```

The JavaBeans standard requires all classes to provide a zero-argument constructor. If, when a JSP-page is resolved, a declared bean does not already exist, it is automatically created using the zero-argument constructor. Furthermore, the JavaBeans standard require all objects to have a public *accessor* and *mutator* method (“getter” and “setter”) for each variable, and that these follow the naming/typing convention that a variable named `firstName` has getter named `getFirstName` and setter and `setFirstName`.

### Example source code of a Java bean

```
package myBeans;

public class User {

    // Public zero-argument constructor
    public User() {

        }

    // JavaBeans variables should always be private
    private String firstName = "";

    // Accessor method ("getter")
    public String getFirstName() {
        return firstName;
    }

    // Mutator method ("setter")
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
}
```

Getter-methods are called when bean properties are presented to the user and setter-methods are called when bean properties are modified through user input.

### Example JSP source code (continued). Presentation and modification of a bean property

```
<jsp:setProperty name="user" property="*" />

<form name="input" action="modifyFirstName.jsp" method="post">
    Example: <input type="text" name="firstName" value="${user.firstName}" size="20" ><br>
    <input type="submit" value="Modify the firstName-property!" />
</form>
```

Instead of merely getting and setting variables straightforward, I exploit some of these required methods to trigger more complex actions in the background. This way, simple user input provided through the JSP interface is able to trigger the actions that are needed in order

to solve the task of presenting users with tailored information about product safety. The JSP layer is used solely as a user interface, leaving all complex tasks for the Java “back-end”.

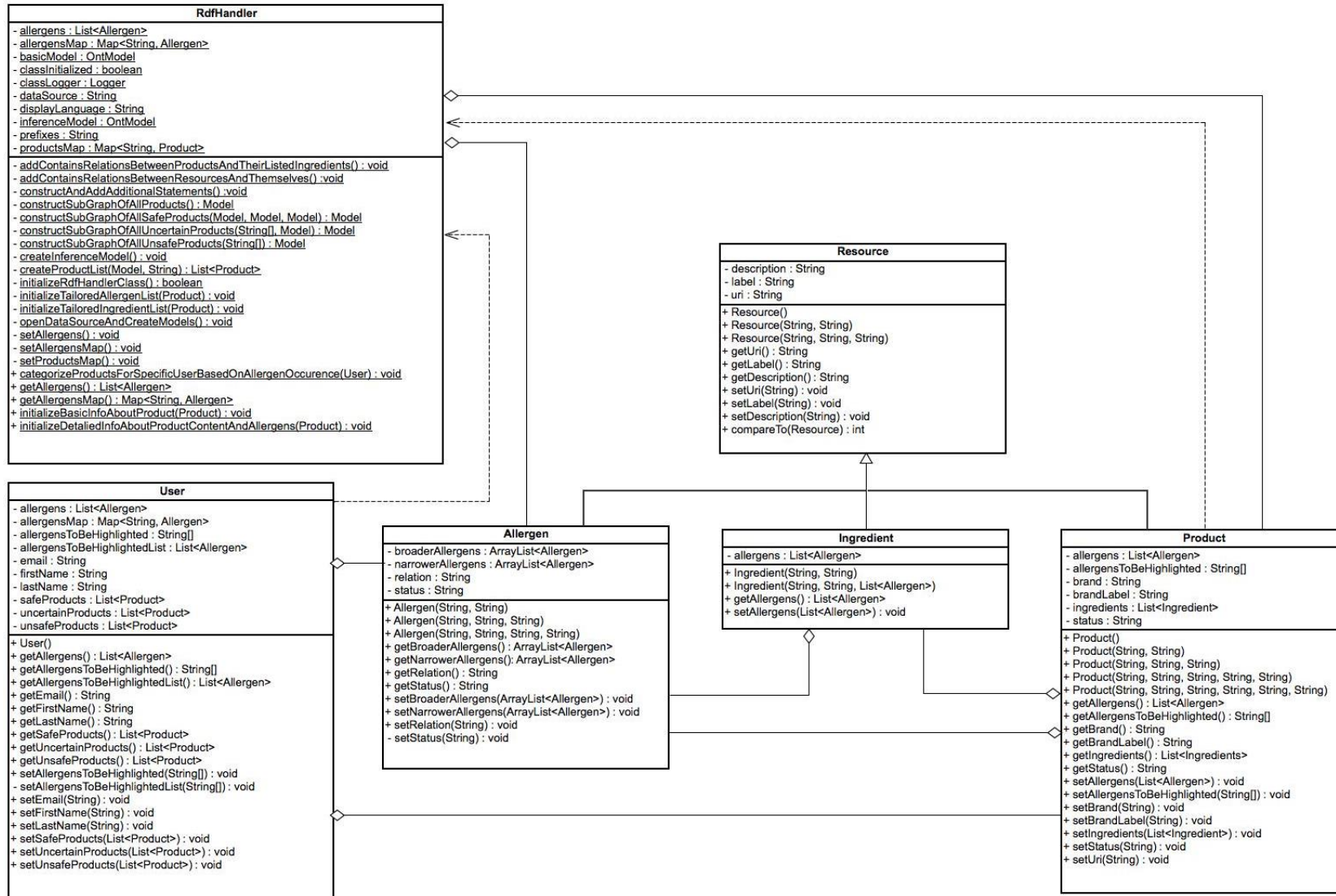
## 6.5 UML Class diagram

The UML class diagram in Figure 31 shows the classes I have developed for this application and the dependencies between them. In addition to these classes, the application relies on several classes from the Jena library and other utility libraries that are not included in the diagram.

In the following, I will briefly introduce the responsibilities of the classes shown in the diagram, before I proceed to explain how the application works, exemplified by a use case expressed as a UML sequence diagram.



Figure 31 UML class diagram



### 6.5.1 The RdfHandler class

The Java class named `RdfHandler` performs all actions that involve dealing with the RDF data directly, and is the core component of the whole application. When a user interacts with the JSP interface, getter and setter methods in the Java beans are called, which in turn make calls to methods in the `RdfHandler` class. These methods perform extensive, tailored SPARQL queries and set operations in order to initialize other bean properties. Observe that the JSP layer never interacts directly with the class; only indirectly by calling getters and setters in the Java beans, which in turn make calls to methods in the `RdfHandler`.

The `RdfHandler` holds two different Jena ontology models: `basicModel` and `inferenceModel`.

Jena `Models` are sets of RDF-statements that, among other things, can be queried using SPARQL. The Jena library provides several different types of models, each with different capabilities. Ontology models (`OntModel`) can be used for data sets that refer to a specified ontology vocabulary. In this case, the data source is an OWL file that holds both ontology and instances, serialized by Protegé. Moreover, when the `basicModel` is initialized with the OWL-file, the asserted statements are supplemented with additional statements – expressing the resources' membership to super classes, according to the specified ontology.

Asserted statement: `Resource R type Allergen`

Ontology statements: `Class Allergen subclassOf Class Substance`  
`Class Substance subclassOf Class Thing`

Inferred statements: `Resource R type Substance`  
`Resource R type Thing`

This extension of the model makes it possible to write simple SPARQL queries that return the expected results. A query for all substances would also return all allergens, even though the allergens' membership to the Substance class was not explicitly expressed in the original dataset, but rather expressed as a relationship between the two classes `Allergen` and `Substance` in the ontology.

The `inferenceModel` is based on the `basicModel`, but the supplied *reasoner* (Apache Jena, n.d.) performs additional inference over the transitive properties (`contains`, `mayContain` etc.). The `inferenceModel` is queried whenever the *inferred statements* are relevant, while the `basicModel` is queried if only *asserted statements* are of interest.

Due to the small scale of the test data set, I have chosen to use Jena Models that are kept “in memory”.

For reasons of efficiency, the `RdfHandler` class performs every operation that can be performed once and for all, independently of individual users’ settings. The `RdfHandler` class is never instantiated, thus all variables and methods are static.

### 6.5.2 The User class

The `User` class is used to store information provided by individual users throughout the HTTP session. When a user enters the web site, a user bean is created. This Java bean is available throughout the remaining session. As the class diagram shows, the user bean holds three lists of `Product` objects: `safeProducts`, `uncertainProducts` and `unsafeProducts`. These lists are initialized by the `RdfHandler` class based on the user’s input, and are used to present the *status* (*safe*, *uncertain* or *unsafe*) of products for the given user.

### 6.5.3 The Resource class

The `Resource` class is a super class used to define common properties of all the objects that originate from the RDF model: products, ingredients and allergens. These objects all have a unique `uri`, a `label` and a `description`. The `Resource` class provides constructors for initializing these variables, as well as setters and getters. It also defines a sorting criterion that makes it possible to order a list of resource objects by their `label`. This is used to present lists of allergens or products in an alphabetical order in the user interface.

#### 6.5.4 The Allergen class

Instances of the `Allergen` class are used to store basic information about allergens. In addition to the properties inherited from the `Resource` super class (`uri`, `label` and `description`), allergen objects may hold a list of narrower allergens and a list of broader allergens. These lists are used to display allergens in a hierarchical way, so that a user may for instance check “nuts” (in general), instead of checking every specific nut-type in the form. The `RdfHandler` class queries the `Jena Model` in order to create a static map of allergen objects. These objects are copied into the users allergen maps upon request, so that it is unnecessary to query the model again for each new user.

#### 6.5.5 The Product class

Instances of this class are used to store information about products. The `RdfHandler` class queries the `Jena Model` in order to obtain a static map of products represented in the dataset. The product map contains `Product` objects that are initialized with basic information such as `uri`, `label`, `brandUri` and `brandLabel`. When a user is presented with lists of safe, uncertain and unsafe products, the product objects in the user bean’s lists are initialized by copying these property values from of the corresponding product objects in the global product map. This is likely to be more efficient than making new SPARQL-queries each time a new user bean needs this basic information. If, however, the user clicks the link to view detailed information about a specific product, or alternatively scans a product directly, a `Product` bean is created and initialized further, with tailored information about the products ingredients and any occurrence the allergens in questions. Since this operation requires several personalized SPARQL-queries to be performed by the `RdfHandler`, it is not carried out until a user actually requests to see this information.

#### 6.5.6 The Ingredient class

Instances of the `Ingredient` class are used to store information about products’ ingredients. `Ingredient` objects are created only when a user actively requests detailed information about a given product. In addition to the basic properties inherited from the super class `Resource` (`uri`, `label` and `description`), `Ingredient` objects hold a list of `Allergen` objects. Only allergens sought by the user in question are kept in the allergen list. Information about

the nature of the relationship between a given ingredient and an allergen (`contains`, `maybeContaminatedBy` etc.) is stored in each allergen object.

## 6.6 UML Sequence diagram

Sequence diagrams can be quite extensive, and in this case, my aim has been to communicate and visualize the overall flow of the program and the interaction between objects and classes that I have developed, without delving into unnecessary details. By grouping together blocks of code into private methods with verbose names that represent the functionality of the method bodies, I was able to create a relatively high-level sequence diagram using a semi-automated sequence diagram generator in Eclipse. For every method displayed in the diagram, I had the option of adding the method calls they lead to. I chose to stop at a quite high abstraction level to make sure that the overall flow wouldn't be lost in unnecessary details.

The sequence diagram in Figure 32 (continued in Figure 36) represents the use case that illustrates the functionality of the program the best: Scan an individual product.

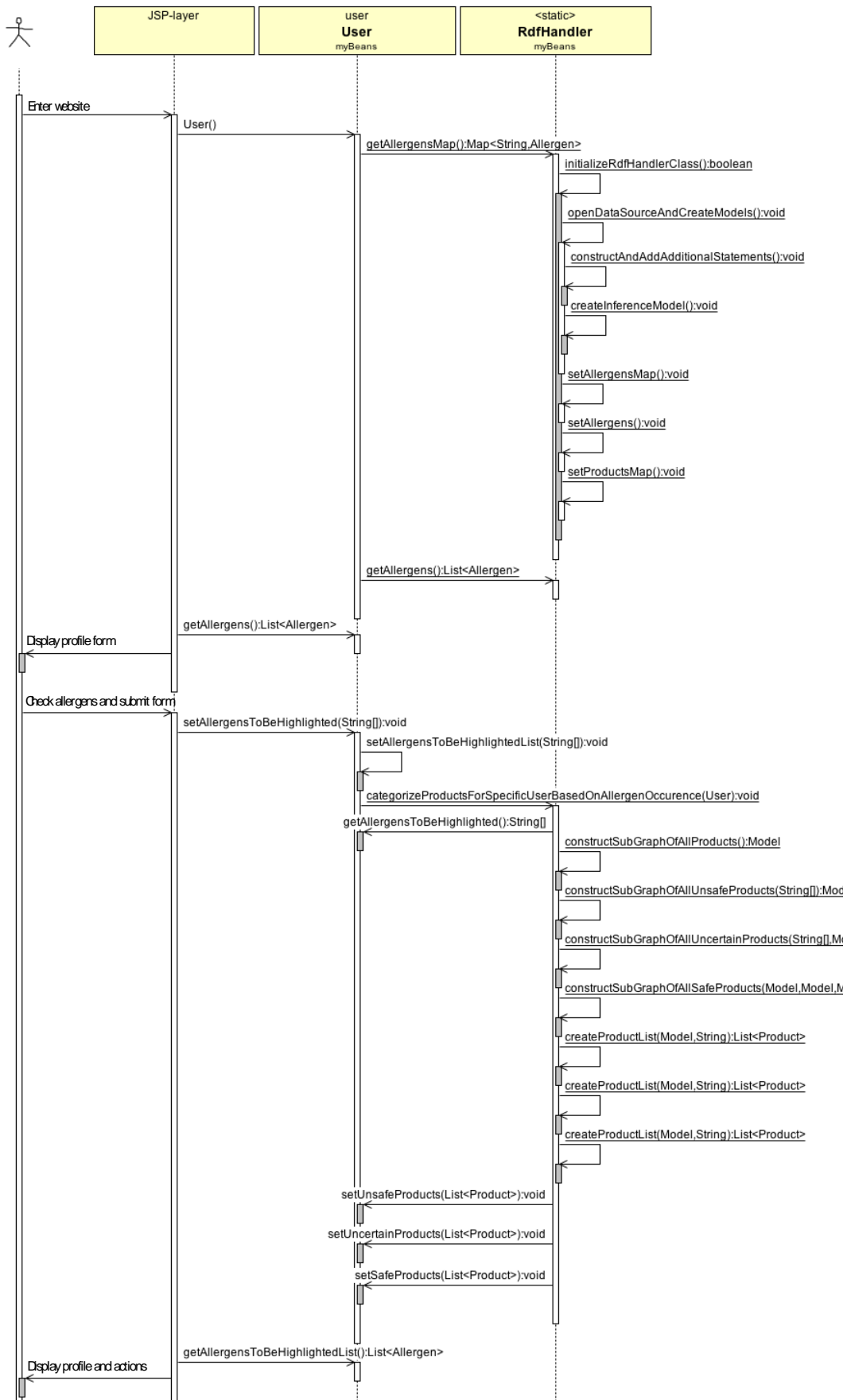
### 6.6.1 Reading the UML sequence diagram

I have split the sequence diagram into two figures for the sake of readability. The yellow rectangles at the top represent classes and objects that play an important role in the execution of the program. The left-most rectangle does not correspond to any particular Java class or object, but it is a simplified representation of the JSP-based user interface, which in itself consists of several different components. The other rectangles correspond to classes or instances of classes already seen in the UML class diagram.

Below each rectangle is a dotted vertical line that changes appearance to indicate that the class or object is active. The sequence diagram is read starting in the upper left corner, following the arrows that represent method calls that are sent between the components. The arrows are labeled with the name of the given method, including indication of any parameters, followed by the methods return value (if any), corresponding to the Java source code.

A thick vertical line represents a running method and the termination of such a line indicates that the given method finishes and sends the specified return value (if any) to the calling place. Moreover, when a running method generates another method call and waits for the result, another thick line is drawn partly overlapping with the existing line.

Figure 32 UML sequence diagram (continued in Figure 36)



## 6.6.2 Walkthrough of the sequence diagram<sup>9</sup>

### A new user enters the website (Ref. use case step 1)

A new user enters the web site after a server restart. The web application is set up to welcome the user with the dynamic web page `home.jsp`. Before the page is served to the user, a number of things happen in the background.

The JSP source code declares the use of a Java bean named `user`, which has `session`-scope.

```
<jsp:useBean id="user" scope="session" class="myBeans.User"/>
```

Since our user is a first-time visitor, there is no pre-existing user bean to be found. Thus a fresh user-bean is created, using the `User` class' zero-argument constructor `User()`.

#### **User()**

The constructor creates a new user object and initializes two of its variables: `allergensMap` and `allergens`. Both represent the `Allergen` resources from the dataset, but their data structures and usage differ. The former is a hash map where `Allergen` objects can be looked up by `uri`. The latter is an ordered list of `Allergen` objects that is used for neat presentation in the JSP-layer. Both variables are initialized with return values from the getter-methods of corresponding `Allergen` objects in the `RdfHandler`'s *static* `allergensMap`.

```
public User() {
    this.allergensMap=RdfHandler.getAllergensMap();
    this.allergens=RdfHandler.getAllergens();
}
```

#### **RdfHandler.getAllergensMap():Map<String, Allergen>**

Normally the `getAllergensMap()` method would simply return the values of the `RdfHandler`'s *static* `allergensMap`, as the method name indicates. However, since our user is the first to use the application after a server restart, the `RdfHandler` class first needs to initialize itself by calling the static method `initializeRdfHandlerClass()`.

---

<sup>9</sup> Setters and getters that don't affect the program flow or perform operations beyond that which their names imply are omitted from the sequence diagram. Some private methods are also omitted from the diagram for the sake of simplicity.



**RdfHandler.initializeRdfHandlerClass():boolean**

The `initializeRdfHandlerClass()` method performs several tasks to prepare the `RdfHandler` to serve the Java beans.

**RdfHandler.openDataSourceAndCreateModels():void**

This method starts by initializing the basic ontology model with the serialized dataset from Protegé, which contains both ontology and instance data.

The ontology model contains explicitly asserted statements with the addition of statements that can be derived from the ontology's definition of super-classes and super-properties. If, for instance, a resource has type `Allergen`, the model automatically adds statements expressing that the resource has type `Substance` and type `Thing`, because these are super-classes of `Allergen`.

The method proceeds to declare all `prefixes` used in the ontology and SPARQL queries once and for all, before it calls a method that algorithmically adds additional statements to the ontology model.

**RdfHandler.constructAndAddAdditionalStatements():void**

This method adds additional statements to the ontology model that are needed to simplify subsequent SPARQL queries. The first part adds direct contains-relations between products and the ingredients in their respective ingredient lists. The second part adds statements that in effect make contains a reflexive property. Both these subtasks are solved by running SPARQL construct queries against the ontology model, thus constructing graphs containing new statements that are subsequently added to the model.

**RdfHandler.createInferenceModel():void**

This method initializes a second ontology model based on the basic ontology model, but with a reasoner that performs additional inference over transitive properties.

### **RdfHandler.setAllergensMap():void**

This method initiates the static map of allergens by querying the basic ontology model.

```
String queryString =
    prefixes +

    " SELECT  ?allergen ?label " +

    " WHERE { " +
    "         ?allergen a ao:Allergen . " +
    "         ?allergen rdfs:label ?label . " +
    "         FILTER (lang (?label) = '" + displayLanguage + "') " +
    "     } " +

    " ORDER BY ASC(?label) ";
```

For each Allergen resource in the model, a Java Allergen object is created. Before the Allergen object is added to the allergensMap, additional queries are made against the basicModel in order to identify and store the allergen's relationship to any broader or narrower allergens (see source code for query details).

### **RdfHandler.setAllergens():void**

This method simply iterates over the aforementioned allergensMap and copies the objects to a list of Allergen objects, which is then sorted by the allergens' labels.

### **RdfHandler.setProductsMap():void**

This method initializes the static map of Product objects by querying the basicModel.

```
String queryString =
    prefixes+

    " SELECT ?product ?label ?brand ?brandLabel " +

    " WHERE { " +
    "         ?product a ao:Product . " +
    "         ?product rdfs:label ?label . " +
    "         ?product ao:hasBrand ?brand . " +
    "         ?brand rdfs:label ?brandLabel . " +
    "     } " +

    "ORDER BY ASC(?label)" ;
```

For each Product resource in the dataset, a Java Product object is created and added to the productsMap with product uri as key.

### **RdfHandler.initializeRdfHandlerClass():Boolean (cont.)**

When all the subtasks of this method has completed, the method returns "true", updating the flag indicating that the class has now been initialized.

**RdfHandler.getAllergensMap() :Map<String, Allergen> (cont.)**

When the initialization of the class is finished, the static `allergensMap` is returned to the calling user object, which copies it into its own corresponding `allergensMap`.

**RdfHandler.getAllergens() :List<Allergen>**

This method simply returns the static ordered list of `Allergen` objects to the calling object, which in turn, copies it into its own corresponding variable.

**User () (cont.)**

The constructor finishes and the new object or bean is available to all the JSP-files in the session-scope.

**user.getAllergens() :List<Allergen>**

This method is called by the `home.jsp` to access the user-bean's list of allergens for presentation in the user interface.

**The system displays the profile form (Ref. use case step 2)**

The dynamic web page is generated and served to the user.

Figure 33 Create profile – Persona E

Welcome

http://localhost:8080/Prototype/hor

Google

Apple Google Maps YouTube Wikipedia Popular (675)

### Create your profile

First name:

Last name:

E-mail address:

### Check everything you want to be warned about

- Celery
- Citrus
- Crustaceans
- Eggs
- Fish
- Gluten
- Legume (all)
- Beans
- Lentils
- Peanuts
- Peas
- Lupine
- Milk (all)
- Lactose
- Milk protein
- Molluscs
- MSG
- Mustard
- Nuts (all)
- Almond
- Hazelnuts
- Peanuts
- Walnuts
- Oats
- Paprika
- Sesame
- Shellfish
- Soy
- Sulfite
- Tomato
- Wheat

### The user checks some allergens and submits the form (Ref. use case step 3)

When the form is submitted, the user bean's array named `allergensToBeHighlighted` is initialized with the URIs of the checked allergens. The URIs are hidden from the user.

```
user.setAllergensToBeHighlighted(String[]  
allergensToBeHighlighted) :void
```

This method first initializes the `allergensToBeHighlighted` array with the URIs of the checked allergens, as the method name indicates. Then a call is sent to the setter-method of `allergensToBeHighlightedList`, which represents the same allergens as an ordered list of `Allergen` objects for presentation in the JSP. Last but not least, a call is sent to the `RdfHandler` method that performs the core functionality of classifying all products in the dataset into the categories `safe`, `uncertain` and `unsafe`, based on the user's input.

```
user.setAllergensToBeHighlightedList(String[]  
allergensToBeHighlighted) : void
```

This method initializes the user bean's `allergensToBeHighlightedList`, which is simply an ordered list of `Allergen` objects corresponding to the allergens checked by the user. This is done by iterating over the `allergensToBeHighlighted` to get hold of the URIs and looking up each URI in the `allergensMap` to access to the corresponding `Allergen` object, which is then added to the list. Finally, the list of allergen objects is sorted according to the specifications given in the `Allergen` class (by label).

```
RdfHandler.categorizeProductsForSpecificUserBasedOnAllergenOccurence  
(User user) : void
```

This method categorizes all products in the dataset into three disjoint categories (safe, uncertain and unsafe) based on the submitted user object's current settings.

SPARQL construct queries are used to construct sub-graphs containing unsafe and uncertain products respectively. The SPARQL queries rely on automatic inference over transitive properties and are thus run against the `inferenceModel`. The safe products sub-graph is not constructed with SPARQL, but instead derived by taking the difference between a graph including all products and the two constructed sub-graphs.

Figure 34 Venn diagrams showing how uncertain and safe products are derived



Safe products  $\cap$  Uncertain products =  $\emptyset$   
 Safe products  $\cap$  Unsafe products =  $\emptyset$   
 Uncertain products  $\cap$  Unsafe products =  $\emptyset$

Unsafe products  $\cup$  Uncertain products  $\cup$  Safe products = All products

**user.getAllergensToBeHighlighted() :String[]**

This method simply gets the checked allergens from the supplied user bean so that `RdfHandler` can loop through the values in the parameterized the SPARQL queries.

**RdfHandler.constructSubGraphOfAllProducts() :Model**

This method constructs a sub-graph of all the products in the `inferenceModel`. It is used later on as a basis to derive the sub-graph of all *safe products* as mentioned above.

**RdfHandler.constructSubGraphOfAllUnsafeProducts(String[] allergensToBeHighlighted) :Model**

This method constructs a sub-graph of all products that are definitely unsafe. That is, the product itself or at least one of its ingredients has a *certain relationship* to (i.e. contains) at least one of the checked allergens (or a broader or narrower form of the allergen). Details of the query are explained in comments underway.

```
// SPARQL CONSTRUCT that generates a sub-graph of all unsafe products in the inference
model

String constructString =
prefixes+

// Construct a graph of products and their respective labels
" CONSTRUCT { " +
"   ?product a ao:Product . " +
"   ?product rdfs:label ?label . " +
" } " +

// That fulfill these conditions:
" WHERE { " +
"   { " +
// The product resources belong to the product class and have a label...
"     ?product a ao:Product . " +
"     ?product rdfs:label ?label . " +

// AND meet at least one of these additional criterion:

// The product contains the allergen
"     { " +
"       ?product ao:contains ?allergen . " +
"     } " +
// OR the product contains a broader variant of the allergen
"     UNION " +
"       { " +
"         ?product ao:contains ?broaderAllergen . " +
"         ?broaderAllergen skos:narrowerTransitive ?allergen . " +
"         ?broaderAllergen a ao:Allergen . " +
"       } " +
// OR the product contains a narrower variant of the allergen
"     UNION " +
"       { " +
"         ?product ao:contains ?narrowerAllergen . " +
"         ?allergen skos:narrowerTransitive ?narrowerAllergen . " +
"         ?narrowerAllergen a ao:Allergen . " +
"       } " +
"     } " +
" } " ;
```

```

// Finally, narrow down the resulting graph to include only matches where the allergen
// URI correspond to an allergen in the submitted array
if (allergensToBeHighlighted.length>0){
    constructString += " FILTER ( ";
    for (int i=0; i < allergensToBeHighlighted.length; i++) {
        constructString += " ?allergen = <" + allergensToBeHighlighted[i] + ">";
        if(i < (allergensToBeHighlighted.length - 1)) {
            constructString += " || " ;
        }
    }
    constructString += " ) ";
}
constructString += " } ";

```

The resulting graph is stored as a Jena model named `unsafeProductsGraph`.

**RdfHandler.constructSubGraphOfAllUncertainProducts (String[]  
allergensToBeHighlighted, Model unsafeProductsGraph) :Model**

This method constructs a sub-graph of all products that are uncertain. That is, the product itself or at least one of its ingredients has an uncertain relationship to at least of the checked allergens (or a broader or narrower form of the allergen). Details of the query are explained in comments underway.

```

// SPARQL CONSTRUCT that generates a sub-graph of all uncertain products in the
inference model

String constructString =
prefixes+

// Construct a graph of products and their respective labels
" CONSTRUCT { " +
"     ?product a ao:Product . " +
"     ?product rdfs:label ?label . " +
" } " +

// That fulfill these conditions:
" WHERE { " +
// The product resources belong to the product class and have a label...
" { " +
"     ?product a ao:Product . " +
"     ?product rdfs:label ?label . " +
" } " +

// AND meet at least one of these additional criterion:
" { " +

// The product has an uncertain relationship to the allergen
"     { " +
"         ?product ao:mayContain ?allergen . " +
"     } " +

// OR the product has an uncertain relationship to a broader or narrower variant of the
allergen
"     UNION " +
"     { " +
"         ?product ao:mayContain ?allergenVariant . " +
"         ?allergen a ao:Allergen . " +
"         { " +
"             { ?allergenVariant skos:narrowerTransitive ?allergen . }
" +
"             UNION " +
"             { ?allergenVariant skos:broaderTransitive ?allergen . } "
+

```



```

"                UNION " +
"                { FILTER ( ?allergenVariant = ?allergen ) } " +
"                } " +
"            } " +

// OR the product has a certain relationship to at least one substance that in turn has
// an uncertain relationship to the allergen
"            UNION " +
"            { " +
"                ?product ao:contains ?substance . " +
"                ?substance ao:mayContain ?allergenVariant ." +
"                ?allergen a ao:Allergen . " +
"                { " +
"                    { ?allergenVariant skos:narrowerTransitive ?allergen . }
"                +
"                    UNION " +
"                    { ?allergenVariant skos:broaderTransitive ?allergen . } "
"                +
"                    UNION " +
"                    { FILTER ( ?allergenVariant = ?allergen ) } " +
"                } " +
"            } " +
"        } " ;

// Finally, narrow down the resulting graph to include only matches where the allergen
// URI correspond to an allergen in the submitted array

if (allergensToBeHighlighted.length>0){
constructString += "FILTER ( ";
    for (int i=0; i < allergensToBeHighlighted.length; i++) {
        constructString += " ?allergen = <" + allergensToBeHighlighted[i] +
"> ";
        if(i < (allergensToBeHighlighted.length - 1)){
            constructString += " || " ;
        }
    }
    constructString += " ) ";
}
constructString += " } " ;

```

The resulting graph is stored as a Jena model named `uncertainProductsGraph`.

Observe that it is quite possible for a product to have a certain relationship to one of the checked allergens and an uncertain relationship to another (or even the same) allergen at the same time. Products like these would match the graph patterns of both the SPARQL constructs just presented, and would thereby be included in both the unsafe and uncertain sub-graphs. Of course, if a product qualifies as both unsafe and uncertain, it should be interpreted as *unsafe*. This is ensured by performing a *difference operation* reinitializing the uncertain product sub-graph to hold all products that were included in the original uncertain products sub-graph *except* the ones that were also included in the unsafe products graph.

```
uncertainProductsGraph=uncertainProductsGraph.difference(unsafeProductsGraph);
```

This operation ensures that the two categories are disjoint.

```
constructSubGraphOfAllSafeProducts (Model allProductsGraph, Model  
unsafeProductsGraph, Model uncertainProductsGraph) :Model
```

This method initializes the sub-graph of safe products by taking the difference between the sub-graph containing all products and the two sub-graphs that were constructed with SPARQL queries.

```
Model safeProductsGraph = allProductsGraph.difference(unsafeProductsGraph);  
safeProductsGraph = safeProductsGraph.difference(uncertainProductsGraph);
```

```
RdfHandler.createProductList (Model graph, String  
status) :List<Product>
```

This method accepts a Jena `Model` and a string indicating whether the model consists of safe, uncertain or unsafe products as parameters. The method queries the submitted Jena `Model` for `Product` resources and converts these into `Product` objects that are put into an ordered list and returned. This method is called three times, once for the graph of safe products, once for the graph of uncertain products and once for the graph of unsafe products.

```
user.setUnsafeProducts (List<Product> unsafeProducts) :void  
user.setUncertainProducts (List<Product> uncertainProducts) :void  
user.setSafeProducts (List<Product> safeProducts) :void
```

These methods set the user-bean's lists of unsafe, uncertain and safe product objects respectively.

When the categorization is finished and the calling user bean's variables has been set the `categorizeProductsForSpecificUserBasedOnAllergenOccurence` method returns.

The calling method `user.setAllergensToBeHighlighted` also returns, leaving control to the JSP layer.

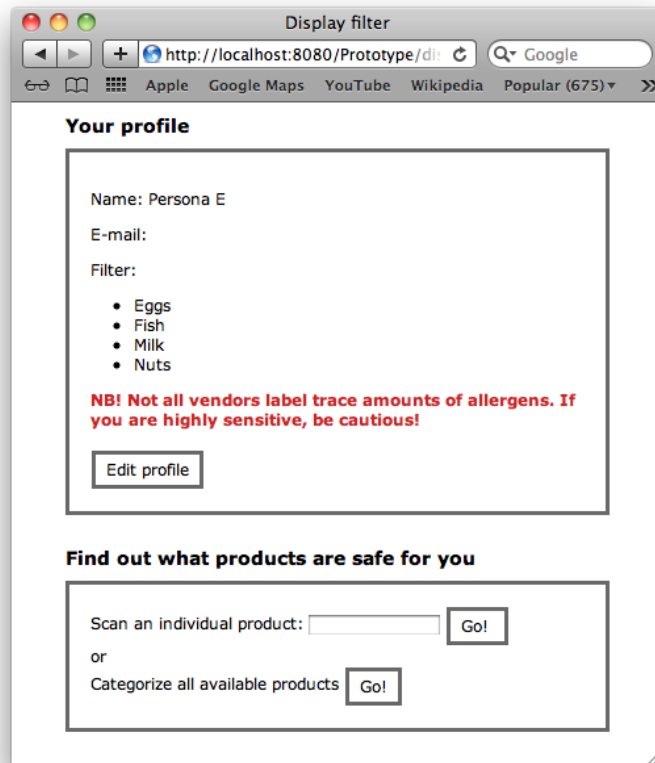
The JSP-layer calls several getters in the user bean in order to generate the next page. Most of these are omitted from the sequence diagram.

```
user.getAllergensToBeHighlightedList () :List<Allergen>
```

This method gets the user beans ordered list of allergens.

The JSP layer displays the user's current profile and available actions (Ref. use case step 4)

Figure 35 Display filter – Persona E



The user scans a product (Ref. use case step 5a)

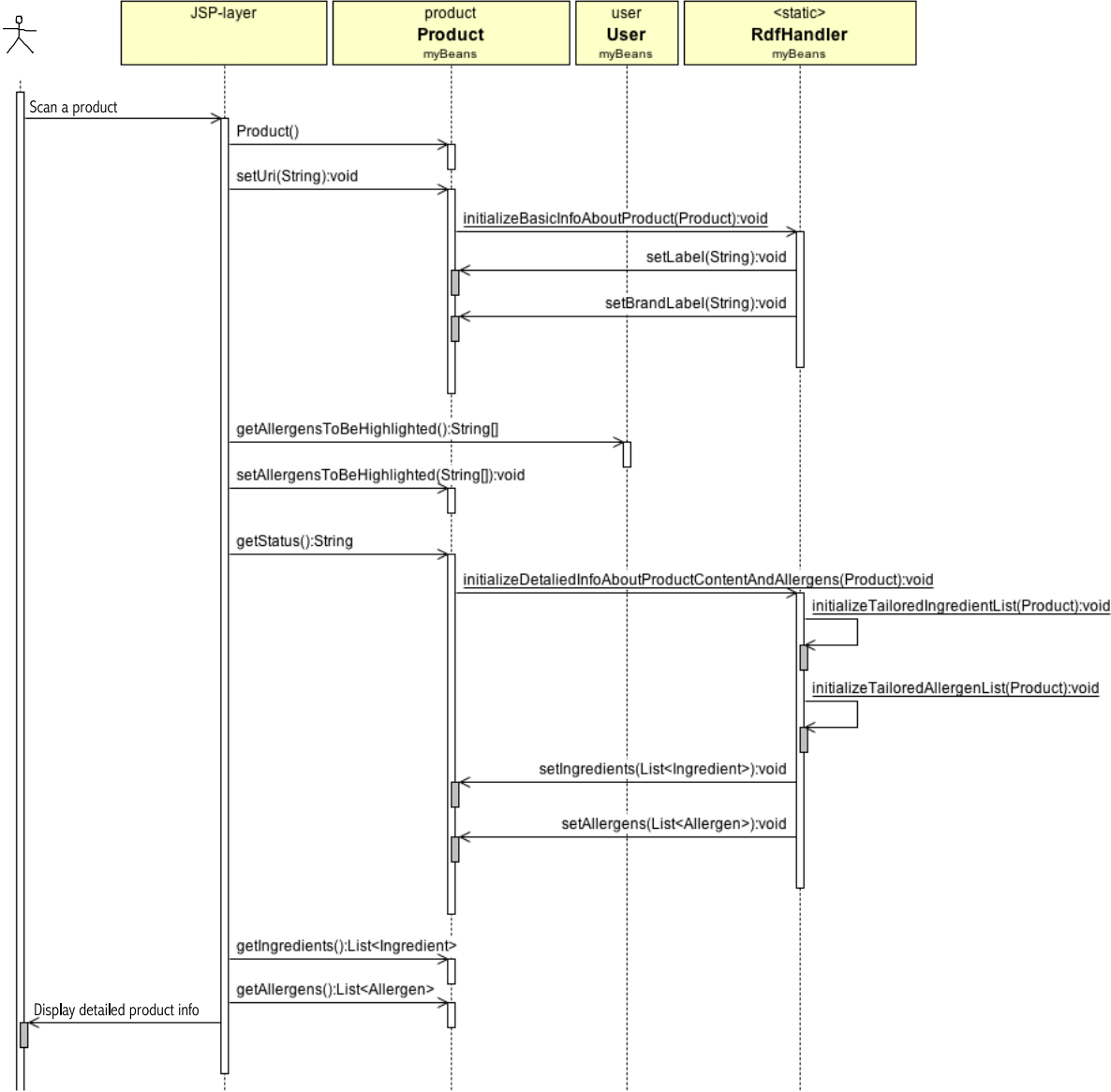
Note that because the prototype is implemented as a web application running in the browser, I have not been able to access the built-in camera found in most smartphones. For now, the barcode scanning is therefore mimicked by manually typing the URI of a product.

When the user submits the URI, the JSP `product_details.jsp` is generated. The page declares use of an additional Java bean. The `product` bean instantiates the `Product` class and has `request` scope.

```
<jsp:useBean id="product" scope="request" class="myBeans.Product"/>
```

This means that the page will be generated again every time the user enters the page. This design choice ensures that the product information will always be presented in accordance with the users current settings.

Figure 36 UML sequence diagram (continued from Figure 32)



**Product ( )**

The constructor calls the super-class Resource’s empty constructor. The returned object’s variables are not initialized.

The JSP declares that the `Product` bean's `uri` property should be initialized with the submitted URI:

```
<jsp:setProperty name="product" property="uri"/>
```

**`product.setUri(String uri):void`**

This method sets the `Product` bean's `uri` property and proceeds to call a method in the `RdfHandler` class that initializes some other basic properties of the bean before it returns.

**`RdfHandler.initializeBasicInfoAboutProduct(Product product):void`**

This method initializes `Product` bean properties that are independent of the user's choice, namely `product label`, `description`, `brandUri` and `brandLabel`. `RdfHandler` looks up the submitted product bean's `uri` in its own static map of products and sets the product bean's properties equal to the property values of the corresponding object in the static map.

I have implemented it this way because the other implemented use case needs fast access to basic information about all products in order to present the lists of safe, uncertain and unsafe products. Since the basic information doesn't need to be adapted to each user's needs, the information can be reused instead of querying the model multiple times for the same information. The compilation of tailored detailed information, on the other hand, is a relatively time consuming process that has to be done for each user and, and is therefore done "on the fly" if and when the user requests it.

The method initializes the product bean's properties by calling its setter methods:

**`product.setLabel(String label):void`**

**`product.setBrandLabel(String brandLabel):void`**

These methods are straightforward setters that don't perform additional operations.

The JSP also declares that the product bean's `allergensToBeHighlighted` property should be set to the value of the calling user bean's corresponding property. This property will later be used to tailor the presentation of the product for the specific user's needs.

```
<jsp:setProperty name="product" property="allergensToBeHighlighted"
value="${user.allergensToBeHighlighted}"/>
```

The JSP calls the user bean's getter to access the array of URIs representing the checked allergens:

```
user.getAllergensToBeHighlighted() :String[]
```

This method is a straightforward getter that doesn't perform any additional operations.

The array of checked allergens is passed on to the product bean's setter:

```
product.setAllergensToBeHighlighted(String[]
allergensToBeHighlighted) :void
```

This method is a straightforward setter that doesn't perform any additional operations.

```
product.getStatus() :String
```

This method returns a string indicating whether the given product is safe, uncertain or unsafe for the current user. The value determines which parts of the CSS will take effect, enhancing the status of the product with green, yellow or red border and heading.

```
<div class="${product.status}" >...</div>
```

The method needs data about which allergens occur in the product in order to determine its safety status. Thus, the method that initializes the detailed information about the product is called.

```
RdfHandler.initializeDetailedInfoAboutProductContentAndAllergens (Pro
duct product) :void
```

This method is responsible for initializing the submitted product bean's lists of ingredients and allergens, according to the user's settings. That is, only allergens that were checked by the user are taken into account.

The task is split into two methods that perform the tasks of initializing the product's `ingredientList` and `allergenList` respectively.

**RdfHandler.initializeTailoredIngredientList(Product product):void**

This method initializes the product bean's ingredientList according to the current filter using SPARQL.

First, all the ingredients in the ingredient list are identified (see SPARQL query below).

```
// SPARQL SELECT query that returns a table of all ingredients in a given product's
ingredient list

String queryString =
prefixes+

" SELECT ?ingredient ?ingredientLabel " +

" WHERE { " +
"   { " +
"     ?product a ao:Product . " +
"     ?product ao:hasIngredientList ?ingredientList . " +
"     ?ingredientList list:member ?ingredient . " +
"     ?ingredient rdfs:label ?ingredientLabel . " +
"   } " +

"   FILTER ( ?product = <" + product.getUri() + "> ) " +
" }";
```

The ingredients are stored as an ordered list of Ingredient objects.

For each ingredient, a new SPARQL query is run in order to identify if they are linked to any of the allergens in question, or any broader or narrower forms of these (see SPARQL query below). The SPARQL query is explained in the comments along the way.

```
// SPARQL SELECT query that returns a table of allergens occurring in an ingredient and
a verbal description of the relationship between the ingredient and each allergen
(contains, may be contaminated by etc.).

String nestedQueryString =
prefixes+

// Select all distinct allergen resources, allergen labels and property labels
" SELECT DISTINCT ?allergen ?allergenLabel ?propertyLabel " +

// That fulfill these conditions:
" WHERE { " +

// The ingredient belongs to the Substance-class, the allergen belongs to the Allergen-
class and both resources have labels
"   ?ingredient a ao:Substance . " +
"   ?allergen a ao:Allergen . " +
"   ?allergen rdfs:label ?allergenLabel . " +
"   ?property rdfs:label ?propertyLabel . " +

// AND meet at least one of these additional criterion:

// The ingredient has a certain relationship to the allergen
"   { " +
"     ?ingredient ?property ?allergen . " +
"     FILTER (?property = ao:contains ) " +
"   } " +
```

```

// OR the ingredient has an uncertain relationship to the allergen (i.e. the property
linking them is a subproperty of may contain)
"    UNION " +
"    { " +
"        ?ingredient ?property ?allergen . " +
"        ?property rdfs:subPropertyOf ao:mayContain . " +
"        FILTER (?property != ao:mayContain ) " +
"    } " +

// OR the ingredient has a certain relationship to a narrower version of the allergen
"    UNION " +
"    { " +
"        ?allergen skos:narrowerTransitive ?narrowerAllergen . " +
"        ?narrowerAllergen a ao:Allergen . " +
"        ?ingredient ?property ?narrowerAllergen . " +
"        FILTER (?property = ao:contains ) " +
"    } " +

// OR the ingredient has an uncertain relationship to to a narrower version of the
allergen
"    UNION " +
"    { " +
"        ?allergen skos:narrowerTransitive ?narrowerAllergen . " +
"        ?narrowerAllergen a ao:Allergen . " +
"        ?ingredient ?property ?narrowerAllergen . " +
"        ?property rdfs:subPropertyOf ao:mayContain . " +
"        FILTER (?property != ao:mayContain ) " +
"    } " +

// OR the ingredient has a certain relationship to a broader version of the allergen
"    UNION " +
"    { " +
"        ?allergen skos:broaderTransitive ?broaderAllergen . " +
"        ?broaderAllergen a ao:Allergen . " +
"        ?ingredient ?property ?broaderAllergen . " +
"        FILTER (?property = ao:contains ) " +
"    } " +

// OR the ingredient has an uncertain relationship to a broader version of the allergen
"    UNION " +
"    { " +
"        ?allergen skos:broaderTransitive ?broaderAllergen . " +
"        ?broaderAllergen a ao:Allergen . " +
"        ?ingredient ?property ?broaderAllergen . " +
"        ?property rdfs:subPropertyOf ao:mayContain . " +
"        FILTER (?property != ao:mayContain ) " +
"    } " +

// Only include results where the ingredient resource has a certain URI
"    FILTER ( ?ingredient = <" + querySolution.getResource("ingredient").toString() + ">
) " ;

// Only include results where the allergen is included in the current filter.
// Iterate array of checked allergens to create filter clause
if (allergensToBeHighlighted.length>0){
    nestedQueryString += "    FILTER ( ";
    for (int i=0; i < allergensToBeHighlighted.length; i++) {
        nestedQueryString +=
" ?allergen = <" + allergensToBeHighlighted[i] + "> " ;
        if(i < (allergensToBeHighlighted.length - 1)){
            nestedQueryString += " || " ;
        }
    }
    nestedQueryString += " ) " ;
}
nestedQueryString += " } " ;

```

The allergens are stored as a list of Allergen objects belonging to the given ingredient. The nature of the relationship between the ingredient and the allergen is stored as a property of the Allergen object.



The method sets the product bean's `ingredientList` and returns.

**`product.setIngredients(List<Ingredient> ingredients):void`**

This method is a straightforward setter that doesn't perform any additional operations.

**`RdfHandler.initializeTailoredAllergenList(Product product):void`**

This method initializes the product bean's `allergenList` according to the current filter using SPARQL.

```
// SPARQL SELECT query that returns a table of allergens that have a directly asserted
relationship to the product and a verbal description of the relationship between the
product and each allergen (produced alongside, may be contaminated by etc.).

String queryString =
prefixes+

// Select all distinct allergen resources, allergen labels and property labels
" SELECT DISTINCT ?allergen ?allergenLabel ?propertyLabel " +

// That fulfill these conditions:
" WHERE { " +

// The product belongs to the Product-class, the allergen belongs to the Allergen-class
and both resources have labels
"   ?product a ao:Product . " +
"   ?allergen a ao:Allergen . " +
"   ?allergen rdfs:label ?allergenLabel . " +
"     ?property rdfs:label ?propertyLabel . " +

// AND meet at least one of these additional criterion:

// The product has an uncertain relationship to the allergen (i.e. the property linking
them is a subproperty of may contain)
"   { " +
"     ?product ?property ?allergen . " +
"     ?property rdfs:subPropertyOf ao:mayContain . " +
"     FILTER (?property != ao:mayContain ) " +
"   } " +

// OR the product has an uncertain relationship to a narrower version of the allergen
" UNION " +
"   { " +
"     ?allergen skos:narrowerTransitive ?narrowerAllergen . " +
"     ?narrowerAllergen a ao:Allergen . " +
"     ?product ?property ?narrowerAllergen . " +
"     ?property rdfs:subPropertyOf ao:mayContain . " +
"     FILTER (?property != ao:mayContain ) " +
"   } " +

// OR the product has an uncertain relationship to a broader version of the allergen
" UNION " +
"   { " +
"     ?allergen skos:broaderTransitive ?broaderAllergen . " +
"     ?broaderAllergen a ao:Allergen . " +
"     ?product ?property ?broaderAllergen . " +
"     ?property rdfs:subPropertyOf ao:mayContain . " +
"     FILTER (?property != ao:mayContain ) " +
"   } " +

// Only include result where the product resource's URI correspond to the product
object's URI
"   FILTER ( ?product = <" + product.getUri() + "> ) " ;
```

```

// Only include results where the allergen is included in the current filter.
if (allergensToBeHighlighted.length>0){
queryString += " FILTER ( ";

    for (int i=0; i < allergensToBeHighlighted.length; i++) {
        queryString +=
" ?allergen = <" + allergensToBeHighlighted[i] + "> " ;
        if(i < (allergensToBeHighlighted.length - 1)){
            queryString += " || " ;
        }
    }

    queryString += " ) ";
}
queryString += "      ]" ;

```

The allergens are stored as an ordered list of `Allergen` objects. The nature of the relationship between the product and the allergen is stored as a property of the allergen object.

The method sets the product bean's `allergenList` and returns.

**`product.setAllergens(List<Allergen> allergens):void`**

This method is a straightforward setter that doesn't perform additional operations.

When the `initializeTailoredIngredientList(Product product)` method returns, the `product.getStatus()` method has the necessary data to determine the correct safety status of the given product for the given user and return it to the JSP layer.

The JSP calls the product bean's getters in order to access the `ingredientList` and `allergenList`.

**`product.getIngredients():List<Ingredient>`**

**`product.getAllergens():List<Allergen>`**

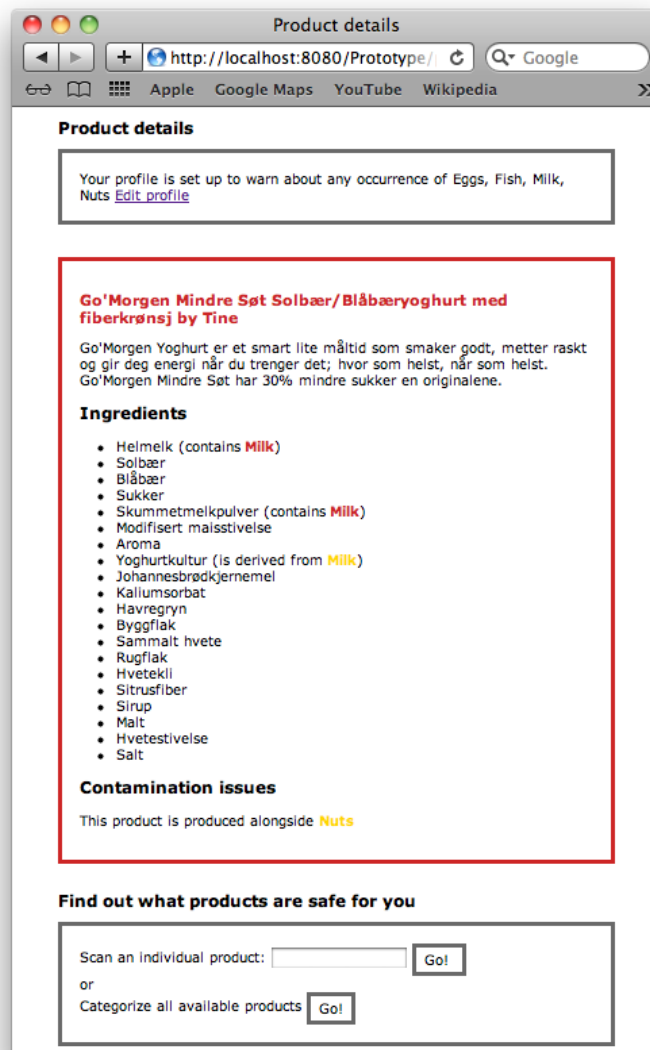
These methods are straightforward getters that don't perform additional operations.

The JSP code iterates over the product bean's `ingredientList` and presents their labels as a HTML list. For each ingredient, it then checks if the ingredient's `allergenList` contains any allergen objects, and if so, these are presented in brackets with CSS-based coloring indicating whether the relationship is of a *certain* (red) or *uncertain* (yellow) nature. The JSP code proceeds to iterate over the product bean's `allergenList` in order to display any

allergen information that is *directly* related to the product (not via its ingredients). The resulting webpage is served to the browser.

**The system displays detailed information about the scanned product (Ref. use case step 6a)**

**Figure 37 Product details – Persona E**



For more details about the implementation, please refer to the attached source code.

## 6.7 Discussion and future work

The prototype is currently based on a small and centralized test dataset. The products in the dataset were handpicked to illustrate particular problems in order to verify that the algorithms would produce the expected results.

### 6.7.1 Optimizing execution time

If a user has checked many allergens (especially allergens that have broader or narrower forms) and proceeds to scan a product that has a very long ingredient list, the execution time to produce the page with detailed product information increases noticeably due to the increased number and complexity of the underlying SPARQL queries.

Extensive logging and timing of the different parts of the algorithm could be performed in order to reveal more information about how the execution time develops as a function of the number of allergens the user has checked and the number of ingredients in the scanned product. Testing the application on real users and a more representative dataset would provide valuable information about how many allergens most users would typically check, and how many ingredients an average product contains, so that the algorithms could be optimized for the most likely scenarios.

### 6.7.2 Enabling distributed data

Moving on from one centralized data source to incorporate multiple distributed data sets wouldn't necessarily require fundamental changes in the application layer, given that the datasets rely on the same ontology and are considered trustworthy.

Each RDF dataset constitutes a graph, and multiple graphs can indeed be merged together to form a new larger graph. The new graph could be serialized as before and used as the data source in the current prototype.

Alternatively, the RDF datasets could be harvested regularly and maintained in a common specialized "triple store". A SPARQL-endpoint could be set up and the application could be modified to run the queries against the SPARQL-endpoint instead of keeping the model in

memory. This solution would require some changes in the application layer, but would probably scale better as the amount of data and number of requests increase.

### **6.7.3 User testing**

The prototype should be tested on real users to see if any changes should be made to make it more usable. For instance, I suspect that the yellow color I have used to communicate that a product is uncertain, might be difficult to read for some. Fortunately, the layered architecture makes it easy to change the presentation layer (JSP and CSS) without changing the underlying algorithms. Moreover, the presentation of the user's current settings and the menu presenting available actions are stored in separate JSP-files that can easily be reused and moved around by use of import statements.

User testing could also determine what use-cases should be developed next, based on the users' expectations to an application like this.

### **6.7.4 Utilizing the smartphone camera as a barcode scanner**

Because the current prototype is implemented as a web application running in the browser, I was unable to utilize the built-in camera found in most smartphones as a scanner. Requiring users to manually type in GTIN codes or search for products by title would be too time consuming in the long run, and users may feel that they are better off reading the textual product declarations. Scanning of product barcodes is an essential feature in order to provide fast feedback about individual products and should therefore be provided in the future implementations of the decision support system.

### **6.7.5 Serialization of user data**

The current prototype does not store users' settings in-between sessions. Food sensitivity is a private matter, so the data should be handled with caution. In the future, the application should provide a secure login feature and offer to serialize user data when the user logs out. When the settings can be stored permanently, users will probably be willing to spend more time and effort setting up their profile. This would allow for more advanced options, such as elimination of particular vendors and specification of which warnings should be taken into

account. Additional services such as a personalized newsletter with tips about safe products and campaigns could also be provided on the basis of the stored user profiles.

## 6.7.6 Refined user profiles

### 6.7.6.1 *Facilitating a more accurate representation users' of dietary needs*

In my prototype, I have sought to enable users to express their needs more accurately by providing an *extended set of allergens* and letting the user chose *specific components* of milk and *specific types* of nuts. The extended list of allergens is inspired by TORO's labeling practice, where selected substances not included in Annex IIIa are labeled with particular care because they are known to cause adverse reactions in some individuals. It would be possible not only to let users "check" various allergens, but to specify whether trace amounts are OK or not for each instance. Persona A would probably check "Allow trace amounts" for gluten while Persona B would check "Zero tolerance" for the same substance. Such an adaptation would make the user interface slightly more complicated, but would enable more precise predictions about food safety for each individual. More products would be classified as either safe or unsafe, reducing the number of uncertain products that the user would need to assess manually.

### 6.7.6.2 *Providing support for other dietary needs*

The ontology could be extended to allow representation of additional product attributes, such as "organic", "fair-trade", "vegetarian", "vegan", "halal" and "kosher". Based on this, the application layer could be adapted to facilitate the needs of other groups with special dietary needs and/or preferences.

However, the proposed model is not suitable for evaluating whether products are "low-fat", "low-carb", "high protein" etc., because this would require *quantification* of ingredients beyond that which vendors are required to submit and subsequent calculation of whether the relative amount of a group of substances is within some ill-defined *threshold*.

### *6.7.6.3 Sub-filters representing the needs of friends and family-members*

The application could be extended to let users operate multiple sub-filters, representing the dietary needs of particular friends and family members. Social features could be provided, enabling people to share their filters with each other. In the shopping situation, users could “check” people who are attending the meal they are planning from a list of “friends”, and have their respective filters combined. This would make it easier for people like Persona C to deal with multiple and changing dietary needs within a group and identify products that all the guests could safely eat.

### **6.7.7 Open source API**

The same core dataset and algorithms could underpin online shops, apps for use in physical stores and vendors’ websites. The application layer containing the algorithms necessary to categorize the products for individual consumers based on their dietary needs could be made available as an open source package that different actors could build on when creating their own services.

## 7 ESTABLISHING THE REQUIRED DATA SETS – DISCUSSION AND FUTURE WORK

In the limited scope of my thesis, my main focus has been to provide proof of concept for the suggested model. In the previous chapters, I have demonstrated that it is possible to utilize Semantic Web standards to create an information system that provides on-demand feedback about product safety for individual people. My prototype relies on a small set of test data consisting of sample products chosen to exemplify some of the issues that the portrayed personas would typically experience. However, if my model were to be used in *real life* and on a *larger scale*, major issues concerning *data acquisition strategies* and *data provenance* would first need to be addressed.

### 7.1 Establishing the authoritative data core

#### 7.1.1 Original strategy – semi-automated conversion of existing data source

When I started working on this project, I remembered having seen an ingredient encyclopedia in booklet format, aimed at people with the most common food allergies. I took for granted that either governmental bodies or stakeholders in the research community would maintain an extensive data bank holding structured data about allergen occurrence in foods. Even though my preliminary research didn't give evidence of any such large-scale data source, my hope was that one would exist for internal use somewhere, and that I could encourage the owner to make the data available for re-use through an open data license.

If this had been the case, it would have been possible to utilize pre-existing tools such as D2RQ (Bizer, Cyganiak, Garbers, Maresch & Becker, 2009) to do a semi-automated conversion of data from existing formats (relational database, XML, etc.) to OWL and RDF. I had made successful use of this approach in a previous project (Holgerson, Preminger & Massey, 2012), and had hoped to build on this experience in the current project. The D2RQ-based approach would involve manual mapping of the data source's underlying data structure to the suggested ontology and subsequent batch conversion of instance data to RDF. However, since most data sources lack sufficient semantics to work outside of its original context, this approach would require a great deal of time-consuming manual proofing.



I tried inquiring both the Norwegian Food Safety Authority (Mattilsynet) by e-mail and the National Register of Severe Allergic Reactions to Food (Matallergiregisteret) in a personal meeting – concerning existing structured data sources that could be used as a basis for the authoritative data core that underpins the suggested model. However, since none of them knew of any structured data sets that could be used for this purpose, I stopped pursuing this approach for the time being.

### 7.1.2 Revised strategy – bottom up, frequency-based population

Since my model relies heavily on semantics and automated deduction, the safest approach would probably be a more controlled “bottom up” strategy, where domain specialists manually assign allergens to every ingredient as it is entered into the data set.

In order to make this task manageable, it would be possible to use ingredient frequency as a basis for prioritizing the order in which ingredients should be added to the model. Some ingredients occur in far more products than others, and some products are sold in much larger quantities than others. By combining these two factors, the occurrence of ingredients could be estimated and plotted. The frequencies would most likely follow a classical “long tail” distribution curve.

It would be feasible to start by adding the high-frequent ingredients from the “head” of the distribution curve into the data set, and then continue systematically in descending order until the majority of ingredients were covered. For the least frequent ingredients it would make more sense to offer vendors a good interface for suggesting new ingredients that are lacking in the data set, than to go through them systematically. An extremely low frequency may indicate that an ingredient is simply a misspelled version of a more common ingredient. The expert team would need to evaluate the suggestions as they come in, and decide in each case whether the new ingredient should be added to the data set or not. If the new ingredient turns out to be synonymous with an existing ingredient, a *same-as* relation should be created between the two resources. The application layer should be adapted to warn users indiscriminately about products where one or more ingredients can’t be identified in the core data set because the expert team hasn’t analyzed them and assigned allergens to them yet.

## 7.2 Selection of allergens

Deciding which semantic entities should be represented in the datasets is a non-trivial process. Biochemists, medical doctors and laymen have different perceptions about what constitutes an *allergen*.

In some cases, scientists have identified particular proteins or parts of proteins that account for the adverse reactions some people experience. Two people allergic to peanuts may even be sensitive towards different proteins occurring in peanuts. For most people, however, distinguishing between the different proteins would have no practical significance. The consequences would be the same in both cases – avoiding peanuts.

In other cases, people experience adverse reactions to substances such as lactose, which is not actually an allergen in the medical sense. However, the practical consequence for a person suffering from severe lactose intolerance is the same as for people suffering from food allergies – eliminating the substance that causes the reaction.

In the dataset developed for the purpose of the prototype, I therefore made a pragmatic choice of assigning all substances that users may want to eliminate due to either food allergy, food intolerance or coeliac disease to the Allergen class.

In the dataset, I have represented allergens as they are listed in Annex IIIa – with the addition of more specific components of milk and specific types of and nuts. I believe that this is a conceptual level that most users would be familiar with and find useful. Inspired by Toro and Gilde, I have included additional allergens that are known to cause allergic reactions, but not currently included in Annex IIIa. The extended list of allergens enables users to provide a more accurate description of their dietary needs to the system.

## 7.3 Semantic distinction between functionally equivalent ingredients

In product declarations, the same ingredient term may refer to several different “things” in the real world. That is, there is no one to one relationship between terms and semantic entities.

### *Examples*

- “Starch” may be derived from different foods such as wheat, corn or potato.
- “Margarine” may be based on either milk fat or vegetable oils.
- “Pesto” is traditionally made with pine nuts, but other types of nuts may occur.

Most people would probably consider the different forms of an ingredient to be equivalent. For people with severe food sensitivities, however, the differences in origin and/or composition may be crucial. In the authoritative data core, generic ingredients like these therefore need to be represented as multiple RDF resources with identical labels. The different resources must be assigned appropriate relationships to allergens individually. RDF descriptions should be used to define the “scope” of each resource, so that vendor representatives composing the product descriptions are able to select the correct RDF resources.

Moreover, vendors may lack sufficient documentation about the exact composition or origin of ingredients bought from suppliers. Different forms of the same ingredient may also be used interchangeably, because they serve the same function in the recipe. In the authoritative data core, each generic ingredient should therefore also be represented with an RDF resource that accounts for this type of uncertainty. All allergens that are known to occur in the various forms of the ingredient should be assigned to this RDF resource. Since end users only see the labels of the RDF resources, the differentiation between nearly equivalent “things” would not result in extra noise.

## **7.4 Data quality and automated inference**

It is important to keep in mind that a Semantic Web based system cannot claim anything about the real world – only what follows from the statements contained in the model. If the data put into the model are of low quality, the automated deductions cannot be trusted. A single error may propagate throughout the model as a consequence of the automatic inference. Measures must thus be taken to reduce the chances of erroneous assertions being added to the data sets and to ensure that any existing errors can be detected and addressed efficiently.

## 7.5 Critical mass of product data

In order to get users to adopt the proposed decision support system, the underlying datasets need to cover the majority of products available. If users scan products only to find that they are not included in the dataset, they will be frustrated and less likely to use the tool again.

## 7.6 Reliable data, inference and algorithms

Reliability is an absolute requirement for the proposed decision support system, since the consequences of giving misinformation about whether a product is safe for an allergic person is severe. If users can't trust the system, it's worthless. Both the datasets and the automated deductions made by the system needs to be trustworthy.

## 7.7 User interface for data providers

It is crucial that the people responsible for the various data sets are provided with a user-friendly and efficient system for entering and maintaining data. Assigning RDF resources with SKOS hidden alternative labels would facilitate identification of resources through text-based search. The system should aid users in selecting the correct RDF resources in cases where several nearly equivalent options are available, such as different variants of “starch” or “margarine”. This could be facilitated by SKOS-based references between related resources, allowing users to browse the model. The user interface should display relevant entailments of statements as they are added to the system. E.g. a vendor representative entering a product declaration, specifying that the product contains a particular ingredient, should be made aware that this implies that the product contains or may contain the allergens that are associated with the given ingredient.

## 7.8 Facilitating efficient feedback from users to data providers

It is important that users are able to report errors they encounter to the data providers in an efficient manner. Both the ontology and application layer should be extended to provide infrastructure to handle such feedback. Product-entries that have been reported as erroneous could be flagged with a warning until the issue has been resolved by the data providers. The warning could momentarily lead the product to be classified as uncertain (at best) for all users.

## 7.9 Dealing with unreliable data providers

Vendors known to provide inadequate or misleading data could be flagged as unreliable either by the Norwegian Food Safety Authority (Mattilsynet) or by patient organizations. The warnings could be transferred to all of the vendor's products through automatic inference, leading the products to be classified as uncertain – regardless of whether the allergens specified by the user have any relationship to the products according to the inference model. Particularly unserious vendors could be blacklisted, leading all of their products to be classified as unsafe or simply eliminated from the data set.

Users could be allowed to specify whether or not they want to take different types of warnings into account. Fewer people would probably be interested in warnings given by e.g. the Norwegian Coeliac Society than the Norwegian Food Safety Authority.

## 7.10 RDF-based certificate specifying labeling practice

Vendors follow different practices regarding allergen warnings. In an ideal world, all vendors could be required to follow the same standard. I believe that this would be unrealistic in practice, unless the standard was set quite low. It is clear that some vendors are more oriented towards customers with special dietary needs than others, and the extra allergen warnings and guarantees they provide should not be withheld for the sake of consistency.

Instead, I propose that individual product declarations should be assigned with a RDF-based certificate, specifying which standard has been followed. The certificate should state which allergens the vendor has committed to warn about.

The lowest permissible standard should require mandatory declaration of all allergens included in Annex IIIa, but no warnings about possible contaminants. Vendors like Toro and Gilde, who have chosen to specify trace amounts of additional substances, would specify this in the individual product certificates – regardless of whether the substances occur or not. This way, a customer who is allergic to e.g. paprika or citrus (which is not included in Annex IIIa) would know for certain whether these product are safe or not. Products from vendors who haven't committed to label trace amounts of the given allergens could contain trace amounts

without it being evident from the product declarations. These products would therefore be classified as uncertain (at best). The user should obviously be informed about the reason why the product has been classified as uncertain; “The vendor has not committed to label trace amounts of X. Trace amounts of X therefore cannot be ruled out.”.

### 7.11 Incentivizing accurate allergen labeling

Both under-use and over-use of allergen warnings is a problem. Since the proposed model is meant to be collaborative, such that different vendors and grocery stores contribute with their own data, it would be possible to allow individual users to eliminate products from vendors they don't trust. Information about which vendors people consider trustworthy or not when it comes to food labeling and unexpected allergen occurrence spreads fast through social media. Conversely, over-use of allergen warnings will make users choose alternative products and thus decrease sales. It's likely that at least the most sensitive people will opt for products classified as safe (green), instead of investigating the specific risk factors associated with a product classified as uncertain (yellow). The proposed model thus reinforces the competitive advantage of providing users with precise and exhaustive allergen information.

### 7.12 Multilingual support

Semantic Web standards are meant to capture the *semantics* of a domain in a way that computers are able to process. However, all RDF classes, properties and instances can be assigned textual labels in order to make the content more comprehensible to people. Literal values can optionally contain a standardized language tag, such as @no (Norwegian) or @en (English). SPARQL queries can filter results by the value of the language tag, thus disregarding labels in all but one language. RDF can therefore be an excellent basis for multilingual applications.

Persona B exemplifies a person who is unable to fully comprehend product descriptions given in Norwegian only. In the development of the prototype application, I therefore wanted to explore the multilingual potential of my proposed model. However, translating the `rdfs:label` of all resources and properties in the test data set would be too time consuming at this point. I therefore made the choice to provide both English and Norwegian labels for all `Allergen` resources and all properties that are rendered in the user interface, while leaving

the language of Ingredient labels unspecified. The unspecified Ingredient labels are all in Norwegian because the dataset was constructed based on products with Norwegian labeling. The screen shots provided in this thesis therefore contain a mixture of Norwegian and English terms (i.e. English Allergen and property labels and Norwegian Ingredient labels.)

The preferred language is currently specified once and for all in the application layer. In the future, users should obviously be allowed to choose which language they prefer.

### **7.13 Multiple application layers based on the same data**

The cooperative model allows individual stakeholders to develop their own application layers, utilizing the authoritative data core in combination with other available data sets. Different applications may provide specialized features directed at their particular target audience. Some examples are features that aid the users' ability to identify and retrieve products by searching and browsing, such as multi-faceted categorization of products with references to the products' location in a specific physical store, references to other available safe products within the same category and full text search for both categories and individual products. Social features such as product ratings and reviews could help users when choosing between similar products. For commercial actors, in-app commercials, campaigns and pricing information could support business goals and increase sales.

## 8 CONCLUSION

In this thesis, I have shown that food sensitivities such as allergies, intolerances and coeliac disease lead to information needs in the shopping situation. Five personas, developed for the purpose of the project, were provided to illustrate the diversity of problematic situations and information needs experienced by people directly or indirectly affected by food sensitivities. Examples of various measures taken to aid these customers in their search for safe foods were given, followed by a discussion of their shortcomings seen from an information science perspective.

An alternative approach based on Semantic Web technologies and Linked Data was proposed. I have argued that public authorities should establish and maintain an authoritative data core, representing knowledge about allergen occurrence in ingredients occurring in foods on the market. This would enable vendors to publish product descriptions as RDF with references to ingredient and allergen resources in the authoritative data core. Using transitive properties from the common domain ontology would reduce the need for manual assignment of allergens to individual products, because this could be automatically inferred from the allergen occurrence in the products' ingredients. Grocery stores would in turn be able to utilize the available datasets to provide customized information services to their customers. The Linked Data approach calls for a comprehensive restructuring of the cooperation between the various players in the food industry.

The proposed model underpins a decision support system that automatically classifies products as safe, uncertain or unsafe for individual users, and communicates the safety status using familiar traffic light colors. Information that is likely to influence the user's manual assessment of uncertain products is emphasized – while other facts are held back to avoid “information overload”. Information loss between the vendor and end user is limited by offering a wide range of properties to express the nature of the relationships between products or ingredients and allergens. An extended and hierarchically structured list of allergens available for selection in the user interface enables users to express their dietary need with greater accuracy, thus enabling better prediction of product safety for each person.



A proof of concept for the decision support system has been provided through a prototype web application. The development process behind the ontology and application layer has been discussed in detail, followed by a discussion of how the required authoritative data core and vendor datasets could be established and quality assured. Suggestions for future work regarding particular aspects of the ontology, application layer and datasets were discussed in the respective chapters.

The prototype application is available for testing at <http://ekko.hioa.no:8090/Prototype/> as of June 2013. The source code is attached at the end of the thesis.

There is still a long way to go before the decision support system could be made available to people with food sensitivities, mainly because the required datasets are not available at this point. However, I hope that my work will inspire industry actors to focus on semantics, standardization and open data in the future, and that emerging information services will be based on a thorough understanding of users' information needs.

## 9 REFERENCE LIST

- Allemang, D., & Hendler, J. (2011). *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL* (2nd ed.). Amsterdam: Elsevier.
- Apache Jena. (n.d.). *Reasoners and rule engines: Jena inference support*. Retrieved June 6, 2013, from <http://jena.apache.org/documentation/inference/>
- Belkin, N. J. (2005). Anomalous State of Knowledge. In K. E. Fisher, S. Erdelez and L. McKechnie (Eds.), *ASIST monograph series: Theories of information behavior* (pp. 44-48). Medford, N.J.: Published for the American Society for Information Science and Technology by Information Today.
- Berners-Lee, T. (2006). *Linked Data*. Retrieved June 6, 2013, from <http://www.w3.org/DesignIssues/LinkedData.html>
- Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The Semantic Web. In *Scientific American Magazine*, May 2001. Retrieved June 6, 2013, from <http://www.scientificamerican.com/article.cfm?id=the-semantic-web>
- Bizer, C., Cyganiak, R., Garbers, J., Maresch, O., Becker, C. (2009). *The D2RQ Platform v0.7: Treating Non-RDF Relational Databases as Virtual RDF Graphs*. [User Manual and Language Specification]. Retrieved June 6, 2013, from <http://www4.wiwiwiss.fu-berlin.de/bizer/d2rq/spec/20090810/>
- Bizer, C., Heath, T., & Berners-Lee, T. (2009). Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 5(3), 1-22.  
doi:10.4018/jswis.2009081901
- Bolle, R. (2012). Reaksjoner på mat – et folkehelseproblem med mange uttrykksformer. *Helserådet rapport*, 20, 3-12.

- Bueso, A. (2012a). Merking av allergener i mat. *Helserådet rapport, 20*, 28-29.
- Bueso, A. (2012b). Nordisk tilsynsrapport viser feilmerking. *Helserådet rapport, 20*, 29-30.
- Coop. (2013). *Nyheter: Coop fjerner gluten fra alle sine panerte svineprodukter* [Press release]. Retrieved May 7, 2013, from <https://coop.no/Om-Coop/Nyheter/?prId=828183&type=pressrelease>
- Druzdzal, M. J., & Flynn, R. R. (2012). *Decision Support Systems*. In M. J. Bates (Ed.), *Understanding information retrieval systems: management, types, and standards* (pp. 461-472). Boca Raton, FL : CRC Press.
- European Parliament, Council. (2000). *Directive 2000/13/EC of the European Parliament and of the Council of 20 March 2000 on the approximation of the laws of the Member States relating to the labeling, presentation and advertising of foodstuffs*. Retrieved April 30, 2013, from <http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:02000L0013-20110120:EN:NOT>
- Forskrift om glutenfrie varer. (2009). *Forskrift om sammensetning og merking av næringsmidler til personer med glutenintoleranse*. Retrieved June 6, 2013, from <http://www.lovdata.no/cgi-wift/ldles?doc=/sf/sf/sf-20090710-0999.html>
- Gilde. (n.d.). *Go' og Mager Leverpostei*. Retrieved May 9, 2013 from <http://www.gilde.no/posteier/go-og-mager-leverpostei-article23930-10190.html>
- Heath, T., & Bizer, C. (2011). *Linked Data: Evolving the Web into a Global Data Space* (1st edition). Morgan & Claypool Publishers.  
doi:10.2200/S00334ED1V01Y201102WBE001
- Hebler, J., Fisher, M., Blace, R., & Perez-Lopez, A. (2009). *Semantic web programming*. Indianapolis, Ind. : Wiley.

Hepp, M. (2011). *GoodRelations Language Reference*. Retrieved June 6, 2013, from <http://purl.org/goodrelations/v1>

Holgersen, R., Preminger, M., & Massey, D. (2012). Using Semantic Web Technologies to Collaboratively Collect and Share User-Generated Content in Order to Enrich the Presentation of Bibliographic Records : Development of a Prototype Based on RDF, D2RQ, Jena, SPARQL and WorldCat's FRBRization Web Service. *Code4Lib Journal* (17). Retrieved June 1, 2013, from <http://journal.code4lib.org/articles/6695>

IFLA Study Group on the Functional Requirements for Bibliographic Records. (2009). *Functional Requirements for Bibliographic Records: Final Report* (Rev. ed.). Retrieved June 7, 2013, from [http://www.ifla.org/files/cataloguing/frbr/frbr\\_2008.pdf](http://www.ifla.org/files/cataloguing/frbr/frbr_2008.pdf)

Internkontrollforskriften for næringsmidler. (1994). *Forskrift om internkontroll for å oppfylle næringsmiddellovgivningen (Internkontrollforskriften for næringsmidler)*. Retrieved June 6, 2013, from <http://www.lovdatab.no/cgi-wift/ldles?doc=/sf/sf/sf-19941215-1187.html>

Jackson, J. C. (2007). *Web Technologies: A Computer Science Perspective*. Upper Saddle River, N.J. : Pearson/Prentice Hall.

Jena: A Semantic Web Framework for Java. (n.d.). Retrieved June 6, 2013, from <http://jena.sourceforge.net/>

Lancaster, F. W. (1991). *Indexing and abstracting in theory and practice*. London: Library Association.

Løvik, M. (2012a). Hvordan bestemmes marking av allergener i mat?. *Helserådet rapport*, 20, 27-28.

Løvik, M. (2012b). Ny mat, nye allergiar. *Helserådet rapport*, 20, 15-16.

- Mathiassen, L., Munk-Madsen, A., Nielsen, P. A., & Stage, J. (2000). *Object Oriented Analysis & Design* (3rd ed.). Ålborg: Marko.
- Matloven. (2003). *Lov om matproduksjon og mattrygghet mv. (matloven)*. Retrieved June 6, 2013, from <http://www.lovdatab.no/all/hl-20031219-124.html>
- Mattilsynet. (2012). *Nordisk tilsynsprosjekt 2010 - 2012 allergenmerking : Norsk rapport*. Retrieved May 2, 2013, from [http://www.mattilsynet.no/mattilsynet/multimedia/archive/00080/Norsk\\_sluttrapport\\_t\\_80153a.pdf](http://www.mattilsynet.no/mattilsynet/multimedia/archive/00080/Norsk_sluttrapport_t_80153a.pdf)
- Mattilsynet. (2013). *Matallergi - merking av matvarer*. Retrieved April 30, 2013, from [http://www.matportalen.no/rad\\_til\\_spesielle\\_grupper/tema/allergikere/matallergi\\_-\\_merking\\_av\\_matvarer](http://www.matportalen.no/rad_til_spesielle_grupper/tema/allergikere/matallergi_-_merking_av_matvarer)
- McBride, B. (2010). *An Introduction to RDF and the Jena RDF API*. Retrieved May 29, 2011, from [http://jena.sourceforge.net/tutorial/RDF\\_API/index.html](http://jena.sourceforge.net/tutorial/RDF_API/index.html)
- Merkeforskriften. (1993). *Forskrift om merking mv av næringsmidler*. Retrieved June 6, 2013 from <http://www.lovdatab.no/cgi-wift/ldles?doc=/sf/sf/sf-19931221-1385.html>
- NAAF. (2013). *CheckContent*. Retrieved June 6, 2013, from <http://www.naaf.no/no/aktuelt/Nyhetsarkiv/CheckContent/>
- NCF. (2013). *App for allergikere*. Retrieved May 10, 2013, from <http://www.ncf.no/Nyheter/App-for-allergikere.aspx>
- NHO Mat og drikke. (2012). *Pressemelding: Allergener* [Press release]. Retrieved May 7, 2013, from <http://www.nhomatogdrikke.no/article.php/category/Pressemeldinger/article/Pressemelding%3A%20Allergener/?articleID=764&categoryID=264>

- Næringsmiddelhygieneforskriften. (2008). *Forskrift om næringsmiddelhygiene (næringsmiddelhygieneforskriften)*. Retrieved June 6, 2013, from <http://www.lovdatab.no/cgi-wift/ldles?doc=/sf/sf/sf-20081222-1623.html>
- CheckContent. (n.d.). Retrieved May 20, 2013 from [www.checkcontent.no](http://www.checkcontent.no)
- Ongstad, P. (1987). *Hva er informasjon?* Transcript of speech presented NSI's info-konferanse, Norway.
- Protégé. (n.d.). *What is protégé?*. Retrieved June 6, 2013, from <http://protege.stanford.edu/overview/>
- Segaran, T., Evans, C., & Taylor, J. (2009). *Programming the Semantic Web*. Sebastopol, Calif. : O'Reilly.
- Senter for sjeldne diagnoser. (2012). *PKU (Fenylketonuri)*. Retrieved May 8, 2013, from <http://www.sjeldnediagnoser.no/?k=PKU%20Fenylketonuri%20&aid=8731>
- Toro. (n.d.). *Om Toro*. Retrieved May 7, 2013, from [http://www.toro.no/om-toro/Om\\_TORO](http://www.toro.no/om-toro/Om_TORO)
- W3C. (2004a). *OWL Web Ontology Language Guide: W3C Recommendation*. Retrieved June 6, 2013, from <http://www.w3.org/TR/2004/REC-owl-guide-20040210/>
- W3C. (2004b). *OWL Web Ontology Language Reference: W3C Recommendation*. Retrieved June 6, 2013, from <http://www.w3.org/TR/owl-ref/>
- W3C. (2004c). *RDF Primer: W3C Recommendation*. Retrieved June 6, 2013, from <http://www.w3.org/TR/2004/REC-rdf-primer-20040210/>
- W3C. (2006). *Defining N-ary Relations on the Semantic Web: W3C Working Group Note*. Retrieved June 6, 2013, from <http://www.w3.org/TR/swbp-n-aryRelations/>

- W3C. (2008). *SPARQL Query Language for RDF: W3C Recommendation*. Retrieved June 6, 2013, from <http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/>
- W3C. (2009a). *SKOS Simple Knowledge Organization System Primer: W3C Working Group Note*. Retrieved June 6, 2013, from <http://www.w3.org/TR/skos-primer/>
- W3C. (2009b). *SKOS Simple Knowledge Organization System Reference: W3C Recommendation*. Retrieved June 6, 2013, from <http://www.w3.org/TR/skos-reference/>
- Wandel, M. (1997). Food labeling from a consumer perspective. *British Food Journal*, 99(6), 212-219. doi:10.1108/00070709710181559
- Wersig, G. (1971). *Information – Kommunikation – Dokumentation*. Pullach bei Munchen: Verlag Dokumentation.

## RdfHandler.java

```
package myBeans;

import java.io.InputStream;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import org.apache.log4j.Logger;
import com.hp.hpl.jena.ontology.OntModel;
import com.hp.hpl.jena.ontology.OntModelSpec;
import com.hp.hpl.jena.query.Query;
import com.hp.hpl.jena.query.QueryExecution;
import com.hp.hpl.jena.query.QueryExecutionFactory;
import com.hp.hpl.jena.query.QueryFactory;
import com.hp.hpl.jena.query.QuerySolution;
import com.hp.hpl.jena.query.ResultSet;
import com.hp.hpl.jena.query.Syntax;
import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.util.FileManager;

public class RdfHandler {

    // Status indicator
    private static boolean classInitialized = false;

    // Logging class
    private static Logger classLogger = Logger.getLogger("myBeans.rdfHandler");

    // Data source - ontology and instances serialized by Protégé
    private static String dataSource = "/home/michaelp/Graph20130516kl0044.owl"; // Linux server
    // private static String dataSource = "C:\\Users\\Michael\\Downloads\\Graph20130516kl0044.owl"; // Windows server
    // private static String dataSource = "/Users/raaghildholgersen/Documents/innlevering/Prototype/Graph20130516kl0044.owl"; // Development environment
    private static String prefixes; // Prefixes used in SPARQL-queries

    // Display language
    private static String displayLanguage = "en"; // "en" or "no" supported

    // Ontology models
    private static OntModel basicModel; // Model containing asserted statements
    private static OntModel inferenceModel; // Same model as above, but with inference.

    // Allergens and Products from data set
    private static Map<String, Allergen> allergensMap; // Used for look-up by URI
```



```

private static List<Allergen> allergens; // Used for alphabetized presentation
private static Map<String, Product> productsMap; // Used for look-up by URI

private static boolean initializeRdfHandlerClass() {
    // This method prepares the class to handle requests from Java Beans

    openDataSourceAndCreateModels();
    setAllergensMap();
    setAllergens();
    setProductsMap();

    // Update the flag to indicate that the class has been initialized
    return classInitialized=true;
}

private static void openDataSourceAndCreateModels() {
    // This method opens the data source (currently a OWL file)

    // Open the data source
    InputStream inputStream = FileManager.get().open(dataSource);
    if (inputStream == null) {
        throw new IllegalArgumentException("File: " + dataSource + " not found");
    }

    // Create an empty Jena ontology model and initialize it with the data set from the OWL-file
    basicModel = ModelFactory.createOntologyModel();
    basicModel.read(inputStream, null);

    // Define prefixes used in the subsequent SPARQL-queries
    prefixes =
        " prefix ao:   <http://www.inferenceTest.owl#> " +
        " prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
        " prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> " +
        " prefix list: <http://jena.hpl.hp.com/ARQ/list#> " +
        " prefix skos: <http://www.w3.org/2004/02/skos/core#> ";

    constructAndAddAdditionalStatements();
    createInferenceModel();
}

private static void constructAndAddAdditionalStatements() {
    // This method adds some additional statements needed to simplify subsequent SPARQL queries

```

```

    addContainsRelationsBetweenProductsAndTheirListedIngredients();
    addContainsRelationsBetweenResourcesAndThemselves();
}

private static void addContainsRelationsBetweenProductsAndTheirListedIngredients() {
    // This method adds direct contains-relations between products and the ingredients in their respective ingredient lists to the basic model

    // SPARQL CONSTRUCT query
    String constructString =
        prefixes +

        " CONSTRUCT { " +
        "   ?product ao:contains ?ingredient . " +
        " } " +

        " WHERE { " +
        "   ?product a ao:Product . " +
        "   ?product ao:hasIngredientList ?ingredientList . " +
        "   ?ingredientList list:member ?ingredient . " +
        " } " ;

    // Create the query
    Query query = QueryFactory.create(constructString, Syntax.syntaxARQ);

    // Execute the query against the basic model
    QueryExecution queryExecution = QueryExecutionFactory.create(query, basicModel);

    // Initialize a new model with the resulting graph
    Model productContainsItsIngredientsStatements = queryExecution.execConstruct();

    // Add the new model to the basic model
    basicModel.add(productContainsItsIngredientsStatements);
}

private static void addContainsRelationsBetweenResourcesAndThemselves() {
    // This method adds statements to the basic model that in effect make contains a reflexive property

    // SPARQL CONSTRUCT query
    String constructString =
        prefixes +

        " CONSTRUCT { " +
        "   ?a ao:contains ?a . " +
        " } " +

```

```

    " WHERE { " +
    "   { ?a a ao:Substance } UNION { ?a a ao:Allergen } " +
    " } " ;

// Create the query
Query query = QueryFactory.create(constructString, Syntax.syntaxARQ);

// Execute the query against the basic model
QueryExecution queryExecution = QueryExecutionFactory.create(query, basicModel);

// Initialize a new model with the resulting graph
Model substancesContainThemSelvesStatements = queryExecution.execConstruct();

// Add the new model to the basic model
basicModel.add(substancesContainThemSelvesStatements);
}

private static void createInferenceModel() {
    // This method initializes a second ontology model based on the basic model, but with a reasoner that performs additional inference

    inferenceModel = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM_MICRO_RULE_INF, basicModel);
    classLogger.debug("OWL_MEM_MICRO_RULE_INF: " + inferenceModel.size() + " ");
}

private static void setAllergensMap() {
    // This method initiates the static map of allergens by querying the basic model

    // The allergen map has URI as key
    allergensMap = new HashMap<String, Allergen>();

    // SPARQL SELECT query
    String queryString =
        prefixes +

        " SELECT DISTINCT ?allergen ?label " +

        " WHERE { " +
        "   ?allergen a ao:Allergen . " +
        "   ?allergen rdfs:label ?label . " +
        "   FILTER (lang (?label) = '" + displayLanguage + "') " +

        " } " +

```

```

    " ORDER BY ASC(?label) ";

// Create the query
Query query = QueryFactory.create(queryString, Syntax.syntaxARQ);

// Execute the query against the basic model
QueryExecution queryExecution = QueryExecutionFactory.create(query, basicModel);

// Put the results into the allergens map
try {
    ResultSet resultSet = queryExecution.execSelect() ;

    while (resultSet.hasNext()) {

        QuerySolution querySolution = resultSet.nextSolution();

        // Create a new allergen object
        Allergen allergen = new Allergen (querySolution.getResource("allergen").toString(), querySolution.getLiteral("label").getValue().toString
(), "Description of the allergen...");

        // SPARQL SELECT query that returns any narrower allergens
        String nestedQueryString =
            prefixes+

            " SELECT ?narrowerAllergen ?label " +

            " WHERE { " +
            "   ?allergen a ao:Allergen . " +
            "   ?narrowerAllergen a ao:Allergen . " +
            "   ?allergen skos:narrowerTransitive ?narrowerAllergen . " +
            "   ?narrowerAllergen rdfs:label ?label . " +
            "   FILTER ( ?allergen = <" + allergen.getUri() + "> ) . " +
            "   FILTER (lang (?label) = '" + displayLanguage + "') " +
            " } " +

            "ORDER BY ASC(?label) ";

        // Create the query
        Query nestedQuery = QueryFactory.create(nestedQueryString, Syntax.syntaxARQ);

        // Execute the query against the inference model
        QueryExecution nestedQueryExecution = QueryExecutionFactory.create(nestedQuery, inferenceModel);

        // Go through the result set
        try {

```

## RdfHandler.java

```

ResultSet nestedResultSet = nestedQueryExecution.execSelect() ;
ArrayList<Allergen> narrowerAllergens = new ArrayList<Allergen>();
while (nestedResultSet.hasNext()) {
    QuerySolution nestedQuerySolution = nestedResultSet.nextSolution();
    // Create a new object representing the narrower allergen and add it to the list
    Allergen narrowerAllergen = new Allergen(nestedQuerySolution.getResource("narrowerAllergen").toString(),
nestedQuerySolution.getLiteral("Label").getValue().toString(), "Description of the allergen...");
    narrowerAllergens.add(narrowerAllergen);
}
// Set the original allergen's list of narrower allergens
allergen.setNarrowerAllergens(narrowerAllergens);
}

finally {
    nestedQueryExecution.close();
}

// SPARQL SELECT query that returns any broader allergens
nestedQueryString =
    prefixes+

    " SELECT ?broaderAllergen ?label " +

    " WHERE { " +
    "   ?allergen a ao:Allergen . " +
    "   ?broaderAllergen a ao:Allergen . " +
    "   ?allergen skos:broaderTransitive ?narrowerAllergen . " +
    "   ?broaderAllergen rdfs:label ?label . " +
    "   FILTER ( ?allergen = <" + allergen.getUri() + "> ) . " +
    "   FILTER (lang (?label) = '" + displayLanguage + "') " +
    " } " +

    "ORDER BY ASC(?label) ";

// Create the query
nestedQuery = QueryFactory.create(nestedQueryString, Syntax.syntaxARQ);

// Execute the query against the inference model
nestedQueryExecution = QueryExecutionFactory.create(nestedQuery, inferenceModel);

// Go through the result set
try {
    ResultSet nestedResultSet = nestedQueryExecution.execSelect();
    ArrayList<Allergen> broaderAllergens = new ArrayList<Allergen>();
    while (nestedResultSet.hasNext()) {

```

RdfHandler.java

```
        QuerySolution nestedQuerySolution = nestedResultSet.nextSolution();
        // Create a new object representing the broader allergen and add it to the list
        Allergen broaderAllergen = new Allergen(nestedQuerySolution.getResource("broaderAllergen").toString(),
nestedQuerySolution.getLiteral("label").getValue().toString(), "Description of the allergen...");
        broaderAllergens.add(broaderAllergen);
    }
    // Set the original allergen's list of broader allergens
    allergen.setBroaderAllergens(broaderAllergens);
}

    finally {
        nestedQueryExecution.close();
    }

    // Add the allergen object to the map of all allergens
    allergensMap.put(allergen.getUri(), allergen);
}
}
finally {
    queryExecution.close();
}
}

public static Map<String, Allergen> getAllergensMap() {
    // This method returns the allergens map

    // Check if the class has been initialized. If not, call the method that does.
    if (!classInitialized) {
        initializeRdfHandlerClass();
    }

    return allergensMap;
}

private static void setAllergens() {
    // This method initializes the ordered list of allergen objects
    allergens = new ArrayList<Allergen>();

    // Go through the allergens map and add the allergen objects to the allergen list
    for (Map.Entry<String, Allergen> entry : allergensMap.entrySet()) {
        allergens.add(entry.getValue());
    }
}
```

## RdfHandler.java

```
// Sort the list of allergen objects according to the specification given in the allergen object (by the allergens' label)
Collections.sort(allergens);
}

public static List<Allergen> getAllergens() {
    // This method returns the list of allergen objects
    return allergens;
}

private static void setProductsMap() {
    // This method initializes the static map of product objects by querying the basic model

    productsMap = new HashMap<String, Product>();

    // SPARQL SELECT query that returns basic information about all products in the basic model
    String queryString =
        prefixes+

        " SELECT ?product ?label ?brand ?brandLabel ?description " +

        " WHERE { " +
        "   { " +
        "     ?product a ao:Product . " +
        "     ?product rdfs:label ?label . " +
        "     ?product ao:hasBrand ?brand . " +
        "     ?brand rdfs:label ?brandLabel . " +
        "     ?product ao:hasProductDescription ?description . " +
        "   } " +
        "   FILTER (BOUND(?description)) " +
        " } " +
        "ORDER BY ASC(?label)" ;

    // Create the query
    Query query = QueryFactory.create(queryString);

    // Execute the query against the basic model
    QueryExecution queryExecution = QueryExecutionFactory.create(query, basicModel);

    // Go through the result set
    try {
        ResultSet results = queryExecution.execSelect() ;

        while (results.hasNext()) {
            QuerySolution querySolution = results.nextSolution() ;
            // Create a new product object and add it to the product map

```

```

        Product p = new Product(querySolution.getResource("product").toString(), querySolution.getLiteral("label").getValue().toString(),
querySolution.getLiteral("description").getValue().toString(), querySolution.getResource("brand").toString(), querySolution.getLiteral
("brandLabel").getValue().toString());
        productsMap.put(p.getUri(), p);
    }
} finally {
    queryExecution.close();
}
}

```

```

public static void categorizeProductsForSpecificUserBasedOnAllergenOccurrence (User user) {
    // This method categorizes all products into three categories (safe, uncertain and unsafe) based on a given user's settings

    // Get the user's settings (i.e. the allergens selected in the JSP-layer
    String[] allergensToBeHighlighted = user.getAllergensToBeHighlighted();

    // Categorize the products for the given user by constructing sub-graphs
    Model allProductsGraph = constructSubGraphOfAllProducts();
    Model unsafeProductsGraph = constructSubGraphOfAllUnsafeProducts(allergensToBeHighlighted);
    Model uncertainProductsGraph = constructSubGraphOfAllUncertainProducts(allergensToBeHighlighted, unsafeProductsGraph );
    Model safeProductsGraph = constructSubGraphOfAllSafeProducts(allProductsGraph, unsafeProductsGraph, uncertainProductsGraph);

    // Create an ordered list for each category
    List<Product> unsafeProducts = createProductList(unsafeProductsGraph, "unsafe");
    List<Product> uncertainProducts = createProductList(uncertainProductsGraph, "uncertain");
    List<Product> safeProducts = createProductList(safeProductsGraph, "safe");

    classLogger.debug("Safe: " + safeProducts.size() + " ");
    classLogger.debug("Uncertain: " + uncertainProducts.size() + " ");
    classLogger.debug("Unsafe: " + unsafeProducts.size() + " ");
    classLogger.debug("Total: " + safeProducts.size()+uncertainProducts.size()+unsafeProducts.size() + " \n");

    // Set the corresponding lists in the calling user bean
    user.setUnsafeProducts(unsafeProducts);
    user.setSafeProducts(safeProducts);
    user.setUncertainProducts(uncertainProducts);
}

```

```

private static Model constructSubGraphOfAllProducts() {
    // This method constructs a sub-graph of all products in the inference model

```



## RdfHandler.java

```
// SPARQL CONSTRUCT query that generates a graph of all products found in the inference model
String constructString =
    prefixes+

    " CONSTRUCT { " +
    "   ?product a ao:Product . " +
    "   ?product rdfs:label ?label . " +
    "   ?product ao:hasBrand ?brand . " +
    " } " +

    " WHERE " +
    " { " +
    "   ?product a ao:Product . " +
    "   ?product rdfs:label ?label . " +
    "   ?product ao:hasBrand ?brand . " +
    " } " ;

// Create the query
Query query = QueryFactory.create(constructString, Syntax.syntaxARQ);

// Execute the query against the inference model and put the resulting graph in a new Jena model
QueryExecution queryExecution = QueryExecutionFactory.create(query, inferenceModel);
Model allProductsGraph = queryExecution.execConstruct();

// Return the sub-graph of all products
return allProductsGraph;
}

private static Model constructSubGraphOfAllUnsafeProducts(String[] allergensToBeHighlighted) {
    if (allergensToBeHighlighted.length==0) {
        Model empty = ModelFactory.createDefaultModel();
        return empty;
    }

    // This method constructs a sub-graph of all products that are unsafe, i.e. the product itself or at least one of its ingredients has a certain
    // relationship to (i.e. contains) one or more of the supplied allergens.

    // SPARQL CONSTRUCT that generates a sub-graph of all unsafe products in the inference model
    String constructString =
        prefixes+

        // Construct a graph of products and their respective labels
```

```

" CONSTRUCT { " +
"   ?product a ao:Product . " +
"   ?product rdfs:label ?label . " +
" } " +

// That fulfill these conditions:
" WHERE { " +
"   { " +
// The product resources belong to the product class and have a label...
"     ?product a ao:Product . " +
"     ?product rdfs:label ?label . " +

// AND meet at least one of these additional criterion:

// The product contains the allergen
"     { " +
"       ?product ao:contains ?allergen . " +
"     } " +
// OR the product contains a broader variant of the allergen
"     UNION " +
"     { " +
"       ?product ao:contains ?broaderAllergen . " +
"       ?broaderAllergen skos:narrowerTransitive ?allergen . " +
"       ?broaderAllergen a ao:Allergen . " +
"     } " +
// OR the product contains a narrower variant of the allergen
"     UNION " +
"     { " +
"       ?product ao:contains ?narrowerAllergen . " +
"       ?allergen skos:narrowerTransitive ?narrowerAllergen . " +
"       ?narrowerAllergen a ao:Allergen . " +
"     } " +
"   } " ;

// Finally, narrow down the resulting graph to include only matches where the allergen URI correspond to an allergen in the submitted array
if (allergensToBeHighlighted.length>0){
  constructString +=
    " FILTER ( ";
  for (int i=0; i < allergensToBeHighlighted.length; i++) {
    constructString += " ?allergen = <" + allergensToBeHighlighted[i] + "> ";
    if(i < (allergensToBeHighlighted.length - 1)) {
      constructString += " || " ;
    }
  }
}
constructString += " ) ";

```

```

}
constructString += " } ";

classLogger.debug("Construct-string: " + constructString + "\n");

// Create the query
Query query = QueryFactory.create(constructString, Syntax.syntaxARQ);

// Execute the query against the inference model and put the resulting sub-graph in a new Jena graph
QueryExecution queryExecution = QueryExecutionFactory.create(query, inferenceModel);
Model unsafeProductsGraph = queryExecution.execConstruct();

// Return the sub-graph of unsafe products
return unsafeProductsGraph;
}

private static Model constructSubGraphOfAllUncertainProducts(String[] allergensToBeHighlighted, Model unsafeProductsGraph) {
    // This method constructs a sub-graph of all products that are uncertain, i.e. the product itself or at least one of its ingredients has an
    uncertain relationship at least of the supplied allergens.

    if (allergensToBeHighlighted.length==0) {
        Model empty = ModelFactory.createDefaultModel();
        return empty;
    }

    // SPARQL CONSTRUCT that generates a sub-graph of all uncertain products in the inference model
    String constructString =
        prefixes+

        // Construct a graph of products and their respective labels
        " CONSTRUCT { " +
        "   ?product a ao:Product . " +
        "   ?product rdfs:label ?label . " +
        " } "
        +
        // That fulfill these conditions:
        " WHERE { " +
        // The product resources belong to the product class and have a label...
        "   { " +
        "     ?product a ao:Product . " +
        "     ?product rdfs:label ?label . " +
        "   } " +

        // AND meet at least one of these additional criterion:
        "   { " +

```

RdfHandler.java

```

// The product has an uncertain relationship to the allergen
"    { " +
"        ?product ao:mayContain ?allergen . " +
"    } " +

// OR the product has an uncertain relationship to a broader or narrower variant of the allergen
"    UNION " +
"    { " +
"        ?product ao:mayContain ?allergenVariant . " +
"        ?allergen a ao:Allergen . "+
"        { " +
"            { ?allergenVariant skos:narrowerTransitive ?allergen . } " +
"            UNION " +
"            { ?allergenVariant skos:broaderTransitive ?allergen . } " +
"            UNION " +
"            { FILTER ( ?allergenVariant = ?allergen ) } " +
"        } " +
"    } " +

// OR the product has a certain relationship to at least one substance that in turn has an uncertain relationship to the allergen
"    UNION " +
"    { " +
"        ?product ao:contains ?substance . " +
"        ?substance ao:mayContain ?allergenVariant . " +
"        ?allergen a ao:Allergen . "+
"        { " +
"            { ?allergenVariant skos:narrowerTransitive ?allergen . } " +
"            UNION " +
"            { ?allergenVariant skos:broaderTransitive ?allergen . } " +
"            UNION " +
"            { FILTER ( ?allergenVariant = ?allergen ) } " +
"        } " +
"    } " +
" } " ;

// Finally, narrow down the resulting graph to include only matches where the allergen URI correspond to an allergen in the submitted array
if (allergensToBeHighlighted.length>0){
    constructString +=
        " FILTER ( ";
    for (int i=0; i < allergensToBeHighlighted.length; i++) {
        constructString += " ?allergen = <" + allergensToBeHighlighted[i] + "> ";
        if(i < (allergensToBeHighlighted.length - 1)){ constructString += " || " ; }
    }
    constructString += " ) ";
}

```

```

}
constructString += " } ";

classLogger.debug("Construct-string: " + constructString);

// Create the query
Query query = QueryFactory.create(constructString, Syntax.syntaxARQ);

// Execute the query against the inference model and put the resulting sub-graph in a new Jena graph
QueryExecution queryExecution = QueryExecutionFactory.create(query, inferenceModel);
Model uncertainProductsGraph = queryExecution.execConstruct();

// Remove all products that are already classified as uncertain
uncertainProductsGraph=uncertainProductsGraph.difference(unsafeProductsGraph);

// Return the sub-graph of uncertain products
return uncertainProductsGraph;
}

private static Model constructSubGraphOfAllSafeProducts(Model allProductsGraph, Model unsafeProductsGraph, Model uncertainProductsGraph) {
    // This method constructs a sub-graph of all products that are safe, i.e. neither the product itself or any of its ingredients have any
    relationship to any of the specified allergens

    // The sub-graph of safe products is found by taking the difference between the graph containing all products and the two graphs containing unsafe
    and uncertain products.
    Model safeProductsGraph = allProductsGraph.difference(unsafeProductsGraph);
    safeProductsGraph = safeProductsGraph.difference(uncertainProductsGraph);

    // Return the sub-graph of safe products
    return safeProductsGraph;
}

private static List<Product> createProductList(Model graph, String status) {
    // This method returns an ordered list of product objects by querying the supplied graph.
    // The status variable indicates whether the supplied graph consists of safe, uncertain or unsafe products.
    // The status is set in each product object, enabling CSS-based formatting communicating products safety with traffic light colors.

    List<Product> productList= new ArrayList<Product>();

    // SPARQL SELECT query that returns all product resources and their labels
    String queryString =
        prefixes +

        " SELECT ?product ?label " +

```

```

    " WHERE { " +
    "   ?product a ao:Product . " +
    "   ?product rdfs:label ?label . " +
    " } " +

    " ORDER BY ASC(?label) " ;

// Create the query
Query query = QueryFactory.create(queryString);

// Execute the query against the supplied graph
QueryExecution queryExecution = QueryExecutionFactory.create(query, graph);

// Go through the result set, create product objects and add them to the product list
try {
    ResultSet results = queryExecution.execSelect() ;

    while (results.hasNext()) {
        QuerySolution querySolution = results.nextSolution() ;
        // Create a new product object and set its status
        Product p = new Product(querySolution.getResource("product").toString(), querySolution.getLiteral("label").getValue().toString()) ;
        p.setStatus(status);
        productList.add(p);
    }

} finally {
    queryExecution.close();
}

// Return the product list
return productList;
}

public static void initializeBasicInfoAboutProduct(Product product) {
    // This method initializes the submitted product object with values from the corresponding product object in the static product map

    if(productsMap.get(product.getUri())!=null) {
        product.setLabel(productsMap.get(product.getUri()).getLabel());
        product.setDescription(productsMap.get(product.getUri()).getDescription());
        product.setBrand(productsMap.get(product.getUri()).getBrand());
        product.setBrandLabel(productsMap.get(product.getUri()).getBrandLabel());
    }
}

public static void initializeDetailedInfoAboutProductContentAndAllergens(Product product) {

```

## RdfHandler.java

```
// This method initializes the submitted product object with detailed information about ingredients and allergen occurrence according to the its current filter
```

```
    initializeTailoredIngredientList(product);  
    initializeTailoredAllergenList(product);  
}
```

```
private static void initializeTailoredIngredientList(Product product) {  
    // This method initializes the product's ingredient list according to the current filter  
  
    // Obtain the product bean's current filter (which again originates from the user bean)  
    String[] allergensToBeHighlighted = product.getAllergensToBeHighlighted();  
  
    // SPARQL SELECT query that returns a table of all ingredients in the product's ingredient list  
    String queryString =  
        prefixes+  
  
        " SELECT ?ingredient ?ingredientLabel " +  
  
        " WHERE {" +  
        "   {" +  
        "     ?product a ao:Product . " +  
        "     ?product ao:hasIngredientList ?ingredientList . " +  
        "     ?ingredientList list:member ?ingredient . " +  
        "     ?ingredient rdfs:label ?ingredientLabel . "+  
        "   } "+  
        "   FILTER ( ?product = <" + product.getUri() + "> ) " +  
        " }" ;  
  
    // Create the query  
    Query query = QueryFactory.create(queryString);  
  
    // Execute the query against the basic model (inference is not needed for this query)  
    QueryExecution queryExecution = QueryExecutionFactory.create(query, basicModel);  
  
    // Make a list for the ingredient objects  
    List<Ingredient> ingredients = new ArrayList<Ingredient>();  
  
    // Go through the ingredients in the result set and find any occurrence of the allergens in question  
    try {  
        ResultSet results = queryExecution.execSelect() ;
```

```

while (results.hasNext()) {
    QuerySolution querySolution = results.nextSolution();

    // SPARQL SELECT query that returns a table of allergens occurring in an ingredient and a verbal description of the relationship between
    the ingredient and each allergen (contains, may be contaminated by etc.).
    String nestedQueryString =
        prefixes+

        // Select all distinct allergen resources, allergen labels and property labels
        " SELECT DISTINCT ?allergen ?allergenLabel ?propertyLabel " +

        // That fulfill these conditions:
        " WHERE { " +

        // The ingredient belongs to the Substance-class, the allergen belongs to the Allergen-class and both resources have labels
        "   ?ingredient a ao:Substance . " +
        "   ?allergen a ao:Allergen . "+
        "   ?allergen rdfs:label ?allergenLabel . " +
        "   ?property rdfs:label ?propertyLabel . " +

        // AND meet at least one of these additional criterion:

        // The ingredient has a certain relationship to the allergen
        "   { " +
        "     ?ingredient ?property ?allergen . " +
        "     FILTER (?property = ao:contains ) " +
        "   } " +

        // OR the ingredient has an uncertain relationship to the allergen (i.e. the property linking them is a subproperty of may contain)
        " UNION "+
        "   { " +
        "     ?ingredient ?property ?allergen . " +
        "     ?property rdfs:subPropertyOf ao:mayContain . " +
        "     FILTER (?property != ao:mayContain ) " +
        "   } " +

        // OR the ingredient has a certain relationship to a narrower version of the allergen
        " UNION "+
        "   { " +
        "     ?allergen skos:narrowerTransitive ?narrowerAllergen . "+
        "     ?narrowerAllergen a ao:Allergen . "+
        "     ?ingredient ?property ?narrowerAllergen . " +
        "     FILTER (?property = ao:contains ) " +
        "   } " +

```



## RdfHandler.java

```

// OR the ingredient has an uncertain relationship to to a narrower version of the allergen
" UNION "+
" { " +
"   ?allergen skos:narrowerTransitive ?narrowerAllergen . "+
"   ?narrowerAllergen a ao:Allergen . "+
"   ?ingredient ?property ?narrowerAllergen . " +
"   ?property rdfs:subPropertyOf ao:mayContain . " +
"   FILTER (?property != ao:mayContain ) " +
" } " +

// OR the ingredient has a certain relationship to a broader version of the allergen
" UNION "+
" { " +
"   ?allergen skos:broaderTransitive ?broaderAllergen . "+
"   ?broaderAllergen a ao:Allergen . "+
"   ?ingredient ?property ?broaderAllergen . " +
"   FILTER (?property = ao:contains ) " +
" } " +

// OR the ingredient has an uncertain relationship to a broader version of the allergen
" UNION "+
" { " +
"   ?allergen skos:broaderTransitive ?broaderAllergen . "+
"   ?broaderAllergen a ao:Allergen . "+
"   ?ingredient ?property ?broaderAllergen . " +
"   ?property rdfs:subPropertyOf ao:mayContain . " +
"   FILTER (?property != ao:mayContain ) " +
" } " +

// Only include results where the ingredient resource has a certain URI
" FILTER ( ?ingredient = <" + querySolution.getResource("ingredient").toString() + "> ) " ;

// Only include results where the allergen is included in the current filter.
// Iterate array of checked allergens to create filter clause
if (allergensToBeHighlighted.length>0){
  nestedQueryString +=
    " FILTER ( ";
  for (int i=0; i < allergensToBeHighlighted.length; i++) {
    nestedQueryString +=
      " ?allergen = <" + allergensToBeHighlighted[i] + "> " ;
    if(i < (allergensToBeHighlighted.length - 1)){
      nestedQueryString += " || " ;
    }
  }
}
}

```

```

        nestedQueryString += " ) ";
    }
    nestedQueryString +=
    "   FILTER (lang (?allergenLabel) = '" + displayLanguage + "') " +
    "   FILTER (lang (?propertyLabel) = '" + displayLanguage + "') ";

    nestedQueryString += " } ";

    // Create the query
    Query nestedQuery = QueryFactory.create(nestedQueryString);

    // Execute the query against the inference model
    QueryExecution nestedQueryExecution = QueryExecutionFactory.create(nestedQuery, inferenceModel);

    // Make a list to store the allergen objects
    List<Allergen> allergens = new ArrayList<Allergen>();

    // Go through the result set, create allergen objects and add them to the list of allergens
    try {
        ResultSet nestedResults = nestedQueryExecution.execSelect() ;

        while (nestedResults.hasNext()) {
            QuerySolution nestedQuerySolution = nestedResults.nextSolution();
            Allergen a = new Allergen(nestedQuerySolution.getResource("allergen").toString(), nestedQuerySolution.getLiteral(
("allergenLabel").getValue().toString(), "Description here...", nestedQuerySolution.getLiteral("propertyLabel").getValue().toString());
            allergens.add(a);
        }

    } finally {
        nestedQueryExecution.close();
    }

    // Create a new ingredient object and initialize it with data obtained from the SPARQL queries
    Ingredient i = new Ingredient(querySolution.getResource("ingredient").toString(), querySolution.getLiteral("ingredientLabel").getValue
().toString(), allergens);

    // Add the new ingredient object to the list
    ingredients.add(i);
}

} finally {
    queryExecution.close();
}

// Set the product's ingredient list

```

```

product.setIngredients(ingredients);

}

private static void initializeTailoredAllergenList(Product product) {
    // This method initializes the product's allergen list based on the current filter

    // Obtain the product bean's current filter (which again originates from the user bean)
    String[] allergensToBeHighlighted = product.getAllergensToBeHighlighted();

    // SPARQL SELECT query that returns a table of allergens that have a directly asserted relationship to the product and a verbal description of the
    relationship between the product and each allergen (produced alongside, may be contaminated by etc.).
    String queryString =
        prefixes+

        // Select all distinct allergen resources, allergen labels and property labels
        " SELECT DISTINCT ?allergen ?allergenLabel ?propertyLabel " +

        // That fulfill these conditions:
        " WHERE { " +

        // The product belongs to the Product-class, the allergen belongs to the Allergen-class and both resources have labels
        " ?product a ao:Product . " +
        " ?allergen a ao:Allergen . " +
        " ?allergen rdfs:label ?allergenLabel . "+
        " ?property rdfs:label ?propertyLabel . " +

        // AND meet at least one of these additional criterion:

        // The product has an uncertain relationship to the allergen (i.e. the property linking them is a subproperty of may contain)
        " { "+
        " ?product ?property ?allergen . " +
        " ?property rdfs:subPropertyOf ao:mayContain . " +
        " FILTER (?property != ao:mayContain ) " +
        " } " +

        // OR the product has an uncertain relationship to a narrower version of the allergen
        " UNION "+
        " { "+
        " ?allergen skos:narrowerTransitive ?narrowerAllergen . "+
        " ?narrowerAllergen a ao:Allergen . "+
        " ?product ?property ?narrowerAllergen . " +
        " ?property rdfs:subPropertyOf ao:mayContain . " +
        " FILTER (?property != ao:mayContain ) " +
        " } " +

```

```

// OR the product has an uncertain relationship to a broader version of the allergen
" UNION "+
" { " +
"   ?allergen skos:broaderTransitive ?broaderAllergen . "+
"   ?broaderAllergen a ao:Allergen . "+
"   ?product ?property ?broaderAllergen . " +
"   ?property rdfs:subPropertyOf ao:mayContain . " +
"   FILTER (?property != ao:mayContain ) " +
" } " +

// Only include result where the product resource's URI correspond to the product object's URI
" FILTER ( ?product = <" + product.getUri() + "> ) " ;

// Only include results where the allergen is included in the current filter.
if (allergensToBeHighlighted.length>0){
  queryString +=
    " FILTER ( ";
  // Iterate array of checked allergens to create filter clause
  for (int i=0; i < allergensToBeHighlighted.length; i++) {
    queryString +=
      " ?allergen = <" + allergensToBeHighlighted[i] + "> " ;
    if(i < (allergensToBeHighlighted.length - 1)){
      queryString += " || " ;
    }
  }
  queryString += " ) ";
}

queryString +=
" FILTER (lang (?allergenLabel) = '" + displayLanguage + "') " +
" FILTER (lang (?propertyLabel) = '" + displayLanguage + "') ";

queryString +=
"   }" ;

// Create the query
Query query = QueryFactory.create(queryString);

// Execute the query against the basic model
QueryExecution queryExecution = QueryExecutionFactory.create(query, basicModel);

// Create a list of allergens
List<Allergen> allergens = new ArrayList<Allergen>();

```

## RdfHandler.java

```
// Go through the result set, create new allergen objects based on the SPARQL query and add them to the allergen list
try {
    ResultSet results = queryExecution.execSelect() ;

    while (results.hasNext()) {
        QuerySolution querySolution = results.nextSolution();
        Allergen a = new Allergen(querySolution.getResource("allergen").toString(), querySolution.getLiteral("allergenLabel").getValue().toString
(), "Description here...", querySolution.getLiteral("propertyLabel").getValue().toString());
        allergens.add(a);
    }

} finally {
    queryExecution.close();
}

// Set the product's allergen list
product.setAllergens(allergens);
}

}
```

## User.java

```
package myBeans;

import java.util.ArrayList;

public class User {

    private Map<String, Allergen> allergensMap = new HashMap<String, Allergen>(); // A map of all allergens originating from the Jena model. Used for
    looking up allergen objects by URI.

    private List<Allergen> allergens = new ArrayList<Allergen>(); // An ordered list of the same allergens, used for presentation in JSP.

    private String firstName = "";
    private String lastName = "";
    private String email = "";

    private String[] allergensToBeHighlighted = new String[0]; // An array of URIs to the allergens selected by the user

    private List<Allergen> allergensToBeHighlightedList = new ArrayList<Allergen>(); // An ordered list of objects representing the same allergens as
    above. Used for presentatin in JSP

    // These maps are initialized with basic data about the products, meant for presentation.
    private List<Product> safeProducts=null;
    private List<Product> uncertainProducts=null;
    private List<Product> unsafeProducts=null;

    public User() {
        this.allergensMap=RdfHandler.getAllergensMap(); // Used for look-up by URI
        this.allergens=RdfHandler.getAllergens(); // Used for ordered presentation
    }

    public List<Allergen> getAllergens() {
        return allergens;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }
}
```

User.java

```
}

public void setLastName(String lastName) {
    this.lastName = lastName;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public void setAllergensToBeHighlighted(String[] allergensToBeHighlighted) {
    // This method triggers the initial categorization of products based on the user's settings

    this.allergensToBeHighlighted=allergensToBeHighlighted;
    if (allergensToBeHighlighted.length>0){
        setAllergensToBeHighlightedList(allergensToBeHighlighted);
    }

    // Request RdfHandler to categorize all products for this user
    RdfHandler.categorizeProductsForSpecificUserBasedOnAllergenOccurrence(this);
}

private void setAllergensToBeHighlightedList(String[] allergensToBeHighlighted) {
    this.allergensToBeHighlightedList = new ArrayList<Allergen>();
    for (int i=0; i< this.allergensToBeHighlighted.length; i++){
        allergensToBeHighlightedList.add((Allergen) allergensMap.get(allergensToBeHighlighted[i]));
    }
    Collections.sort(allergensToBeHighlightedList);
}

public String[] getAllergensToBeHighlighted() { // Used by RdfHandler in tailored SPARQL queries
    return this.allergensToBeHighlighted;
}

public List<Allergen> getAllergensToBeHighlightedList() { // Used by JSP User-bean to retrieve a Map of all allergens that were selected by the user
    if (allergensToBeHighlighted.length<1){ // Nothing was sent with the form...
        setAllergensToBeHighlighted(allergensToBeHighlighted);
    }
    return allergensToBeHighlightedList;
}
```

User.java

```
public void setSafeProducts(List<Product> safeProducts) {
    this.safeProducts = safeProducts;
}

public List<Product> getSafeProducts() {
    return safeProducts;
}

public void setUncertainProducts(List<Product> uncertainProducts) {
    this.uncertainProducts = uncertainProducts;
}

public List<Product> getUncertainProducts() {
    return uncertainProducts;
}

public void setUnsafeProducts(List<Product> unsafeProducts) {
    this.unsafeProducts = unsafeProducts;
}

public List<Product> getUnsafeProducts() {
    return unsafeProducts;
}
}
```



```
package myBeans;

public class Resource implements Comparable<Resource> {

    private String uri;
    private String label;
    private String description;

    public Resource() {
        super();
    }

    public Resource(String uri, String label, String description) {
        super();
        this.uri = uri;
        this.label = label;
        this.description = description;
    }

    public Resource(String uri, String label) {
        super();
        this.uri = uri;
        this.label = label;
    }

    public String getUri() {
        return uri;
    }

    public void setUri(String uri) {
        this.uri = uri;
    }

    public String getLabel() {
        return label;
    }

    public void setLabel(String label) {
        this.label = label;
    }

    @Override
    public int compareTo(Resource r) {
        // Sort objects by label
        return this.label.compareToIgnoreCase(r.getLabel());
    }

    public String getDescription() {
        return description;
    }
}
```

Resource.java

```
public void setDescription(String description) {  
    this.description = description;  
}  
}
```

## Allergen.java

```
package myBeans;

import java.util.ArrayList;

public class Allergen extends Resource {

    private ArrayList<Allergen> narrowerAllergens = new ArrayList<Allergen>();
    private ArrayList<Allergen> broaderAllergens = new ArrayList<Allergen>();
    private String relation = null;
    private String status = null;

    public Allergen(String uri, String label) {
        super(uri, label);
    }

    public Allergen(String uri, String label, String description) {
        super(uri, label, description);
    }

    public Allergen(String uri, String label, String description, String relation) {
        super(uri, label, description);
        this.setRelation(relation);
    }

    public void setNarrowerAllergens(ArrayList<Allergen> narrowerAllergens) {
        this.narrowerAllergens = narrowerAllergens;
    }

    public ArrayList<Allergen> getNarrowerAllergens() {
        return narrowerAllergens;
    }

    public void setBroaderAllergens(ArrayList<Allergen> broaderAllergens) {
        this.broaderAllergens = broaderAllergens;
    }

    public ArrayList<Allergen> getBroaderAllergens() {
        return broaderAllergens;
    }

    public String getRelation() {
        return relation;
    }

    public void setRelation(String relation) {
```

## Allergen.java

```
    this.relation = relation;
    if (relation.equals("contains")){
        this.setStatus("unsafe");
    }
    else
    {
        this.setStatus("uncertain");
    }
}

private void setStatus(String status) {
    this.status=status;
}

public String getStatus() {
    return status;
}
}
```

## Ingredient.java

```
package myBeans;

import java.util.List;

public class Ingredient extends Resource {

    private List<Allergen> allergens;

    public Ingredient(String uri, String label, List<Allergen> allergens) {
        super(uri, label);
        this.allergens=allergens;
    }

    public List<Allergen> getAllergens() {
        return allergens;
    }

    public void setAllergens(List<Allergen> allergens) {
        this.allergens = allergens;
    }
}
```

## Product.java

```
package myBeans;

import java.util.List;

public class Product extends Resource{

    private String brand;
    private String brandLabel;

    private String status="unknown";

    private List<Ingredient> ingredients; // Product's ingredients
    private List<Allergen> allergens; // Product's allergens, incl. data about their relationship to the product

    private String[] allergensToBeHighlighted; // Used to customize presentation according to a specific user's filter

    public Product() {
        super();
    }
    public Product(String uri, String label) {
        super(uri, label);
    }

    public Product(String uri, String label, String description, String brand, String brandLabel) {
        super(uri, label, description);
        this.brand = brand;
        this.brandLabel = brandLabel;
    }

    public void setUri(String uri) {
        super.setUri(uri);
        RdfHandler.initializeBasicInfoAboutProduct(this);
    }

    public String getBrand() {
        return brand;
    }

    public void setBrand(String brand) {
        this.brand = brand;
    }

    public String getBrandLabel() {
        if(brandLabel==null){
```

Product.java

```
        RdfHandler.initializeBasicInfoAboutProduct(this);
    }
    return brandLabel;
}

public void setBrandLabel(String brandLabel) {
    this.brandLabel = brandLabel;
}

public String[] getAllergensToBeHighlighted() {
    return allergensToBeHighlighted;
}

public void setAllergensToBeHighlighted(String[] allergensToBeHighlighted) {
    this.allergensToBeHighlighted = allergensToBeHighlighted;
}

public void setAllergens(List<Allergen> allergens) {
    this.allergens = allergens;
}

public List<Allergen> getAllergens() {
    if (allergens==null){
        RdfHandler.initializeDetailedInfoAboutProductContentAndAllergens(this);
    }
    return allergens;
}

public void setIngredients(List<Ingredient> ingredients) {
    this.ingredients=ingredients;
}

public List<Ingredient> getIngredients() {
    if (ingredients==null){
        RdfHandler.initializeDetailedInfoAboutProductContentAndAllergens(this);
    }
    return ingredients;
}

public String getStatus() {
    if (status=="unknown"){
        RdfHandler.initializeDetailedInfoAboutProductContentAndAllergens(this);
        for(Allergen a : allergens) {
            if (a.getRelation().equals("contains")){
                return status="unsafe";
            }
        }
    }
}
```

Product.java

```
    }
    else {
        status="uncertain";
    }
}
for(Ingredient i : ingredients) {
    for(Allergen a : i.getAllergens()) {
        if (a.getRelation().equals("contains")){
            return status="unsafe";
        }
        else {
            status="uncertain";
        }
    }
}
}
if (status.equals("unknown")){
    return "safe";
}
return status;
}

public void setStatus(String status) {
    this.status = status;
}
}
```



home.jsp

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<%@ page import="java.util.*" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
<meta http-equiv="Pragma" content="no-cache"/>
<meta http-equiv="Cache-Control" content="no-cache"/>
<meta http-equiv="Expires" content="Sat, 01 Dec 2001 00:00:00 GMT"/>

<link rel="stylesheet" href="{pageContext.request.contextPath}/style.css" />

<title>Welcome</title>

</head>

<body>

<jsp:useBean id="user" scope="session" class="myBeans.User"/>
<jsp:setProperty name="user" property="*/>

<h1>Create your profile</h1>

    <form name="input" action="display_filter.jsp" method="post">
    <table>
    <tr><td>First name: </td><td><input type="text" name="firstName" size="50" value="{user.firstName}"/></td></tr>
    <tr><td>Last name: </td><td><input type="text" name="lastName" size="50" value="{user.lastName}"/></td></tr>
    <tr><td>E-mail address: </td><td><input type="text" name="email" size="50" value="{user.email}"/></td></tr>
    </table>
    <h2>Check everything you want to be warned about </h2>

    <c:forEach items="{user.allergens}" var="current" >

    <c:if test="{empty current.broaderAllergens}">
    <input type="checkbox" id="allergensToBeHighlighted" name="allergensToBeHighlighted" value="{current.uri}"
    <c:forEach items="{user.allergensToBeHighlighted}" var="allergenToBeHighlighted" >
    <c:if test="{current.uri == allergenToBeHighlighted}">checked</c:if>
    </c:forEach>
    </c:if>
    </c:forEach>
    </form>
```

home.jsp

```
</c:forEach>
/>
<label for="allergensToBeHighlighted"><c:out value="${current.label}"/><c:if test="${!empty current.narrowerAllergens}"> (all)</c:if></label><br/>

<c:forEach items="${current.narrowerAllergens}" var="item">
<input type="checkbox" id="allergensToBeHighlighted" name="allergensToBeHighlighted" value="${item.uri}"
<c:forEach items="${user.allergensToBeHighlighted}" var="allergenToBeHighlighted" >
<c:if test="${item.uri == allergenToBeHighlighted}">checked</c:if>
</c:forEach>
/>
<label for="allergensToBeHighlighted">--- <c:out value="${item.label}"/></label><br/>
</c:forEach>
</c:if>
</c:forEach>
<br/>

<input type="submit" value="Create profile!" class="button"/>
</form>

<br/>
</body>
</html>
```

## display\_filter.jsp

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<%@ page import="java.util.*" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
<meta http-equiv="Pragma" content="no-cache" />
<meta http-equiv="Cache-Control" content="no-cache" />
<meta http-equiv="Expires" content="Sat, 01 Dec 2001 00:00:00 GMT" />

<link rel="stylesheet" href="${pageContext.request.contextPath}/style.css" />

<title>Display filter</title>

</head>

<body>

    <jsp:useBean id="user" scope="session" class="myBeans.User"/>
    <jsp:setProperty name="user" property="*" />

    <h1>Your profile</h1>
    <div class="filter">
    <p>Name: <jsp:getProperty name="user" property="firstName" />
    <jsp:getProperty name="user" property="lastName" /></p>
    <p>E-mail: <jsp:getProperty name="user" property="email" /> </p>
    <p>Filter: <c:if test="${empty user.allergensToBeHighlightedList}">Your filter is not set. </c:if></p>
    <ul>
    <c:forEach items="${user.allergensToBeHighlightedList}" var="item">
    <li><c:out value="${item.label}"/><br/></li>
    </c:forEach>
    </ul>
    <span class="unsafe">NB! Not all vendors label trace amounts of allergens. If you are highly sensitive, be cautious!</span><br/><br/>
    <form action="home.jsp" >
    <input type="submit" value="Edit profile" tabindex="1" class="button"/>
    </form>
    </div>
    <br/>

```

display\_filter.jsp

```
<jsp:include page="available_actions.jsp"/>  
  
</body>  
</html>
```

available\_actions.jsp

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<%@ page import="java.util.*" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
<meta http-equiv="Pragma" content="no-cache"/>
<meta http-equiv="Cache-Control" content="no-cache"/>
<meta http-equiv="Expires" content="Sat, 01 Dec 2001 00:00:00 GMT"/>

<link rel="stylesheet" href="{pageContext.request.contextPath}/style.css" />

<title>Available actions</title>

</head>

<body>
  <jsp:useBean id="user" scope="session" class="myBeans.User"/>
  <jsp:setProperty name="user" property="*" />
  <h1>Find out what products are safe for you</h1>
  <div class="filter">
    <form action="product_details.jsp" >
      Scan an individual product:
      <input type="text" name="uri"/>

      <input type="submit" value="Go! " tabindex="2" class="button"/>
    </form>
    or
    <form action="categorized_products.jsp" >
      Categorize all available products
      <input type="submit" value="Go!" tabindex="2" class="button" />
    </form>
  </div>
</body>
</html>
```

## filter.jsp

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<%@ page import="java.util.*" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />

</head>

<body>

<jsp:useBean id="user" scope="session" class="myBeans.User"/>

<div class="filter">
<c:if test="${!empty user.firstName && !empty user.lastName}">
Hi, <c:out value="${user.firstName}"/> <c:out value="${user.lastName}"/>!<br/>
</c:if>

<c:if test="${!empty user.allergensToBeHighlightedList}">
    Your profile is set up to warn about any occurrence of
    <c:forEach items="${user.allergensToBeHighlightedList}" var="item" varStatus="status">
        <c:out value="${item.label}"/><c:if test="${not status.last}">, </c:if>
    </c:forEach>
    <a href="home.jsp">Edit profile</a>
</c:if>
<c:if test="${empty user.allergensToBeHighlightedList}">
    For a tailored experience, please <a href="home.jsp">create a filter</a>!
</c:if>
</div>
<br/>
</body>
</html>
```

categorized\_products.jsp

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<%@ page import="java.util.*" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
<meta http-equiv="Pragma" content="no-cache"/>
<meta http-equiv="Cache-Control" content="no-cache"/>
<meta http-equiv="Expires" content="Sat, 01 Dec 2001 00:00:00 GMT"/>

<link rel="stylesheet" href="${pageContext.request.contextPath}/style.css" />

<title>Categorized products</title>

</head>

<body>

<jsp:useBean id="user" scope="session" class="myBeans.User"/>

<h1>Categorized products</h1>

<jsp:include page="filter.jsp"/>

<c:if test="${not empty user.safeProducts}">
<h2>Safe products</h2>
<div class="safe">
  <ul>
    <c:forEach items="${user.safeProducts}" var="item" >
      <li>
        <form name="input" action="product_details.jsp" method="get" >
          <c:out value="${item.label}"/> by <c:out value="${item.brandLabel}"/>
          <input type="hidden" name="uri" value="<c:out value='${item.uri}'/>"/>
          <input type="hidden" name="status" value="<c:out value='${item.status}'/>"/>
          <input type="submit" value="View details" />
        </form>
      </li>
    </c:forEach>
  </div>
</div>
```

```

    </ul>
</div>
<br/>
</c:if>

<c:if test="${not empty user.uncertainProducts}">
<h2>Uncertain products</h2>
<div class="uncertain">
    <ul>
        <c:forEach items="${user.uncertainProducts}" var="item" >
            <li>
                <form name="input" action="product_details.jsp" method="get" >
                    <c:out value="${item.label}"/> by <c:out value="${item.brandLabel}"/>
                    <input type="hidden" name="uri" value="<c:out value='${item.uri}'/>"/>
                    <input type="hidden" name="status" value="<c:out value='${item.status}'/>"/>
                    <input type="submit" value="View details" />
                </form>
            </li>
        </c:forEach>
    </ul>
</div>
<br/>
</c:if>

<c:if test="${not empty user.unsafeProducts}">
<h2>Unsafe products</h2>
<div class="unsafe">
    <ul>
        <c:forEach items="${user.unsafeProducts}" var="item" >
            <li>
                <form name="input" action="product_details.jsp" method="get" >
                    <c:out value="${item.label}"/> by <c:out value="${item.brandLabel}"/>
                    <input type="hidden" name="uri" value="<c:out value='${item.uri}'/>"/>
                    <input type="hidden" name="status" value="<c:out value='${item.status}'/>"/>
                    <input type="submit" value="View details" />
                </form>
            </li>
        </c:forEach>
    </ul>
</div>
<br/>
</c:if>

<br/>

```



categorized\_products.jsp

```
</body>  
</html>
```

product\_details.jsp

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<%@ page import="java.util.*" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
<meta http-equiv="Pragma" content="no-cache">
<meta http-equiv="Cache-Control" content="no-cache">
<meta http-equiv="Expires" content="Sat, 01 Dec 2001 00:00:00 GMT">

<link rel="stylesheet" href="{pageContext.request.contextPath}/style.css" />

<title>Product details</title>

</head>

<body>
<jsp:useBean id="user" scope="session" class="myBeans.User"/>
<jsp:setProperty name="user" property="*" />
<jsp:useBean id="product" scope="request" class="myBeans.Product"/>
<jsp:setProperty name="product" property="uri" />
<jsp:setProperty name="product" property="allergensToBeHighlighted" value="{user.allergensToBeHighlighted}" />

<h1>Product details</h1>
<jsp:include page="filter.jsp" />
<br />

<c:if test="{!empty product.label}">
<div class="{product.status}" >
<h2 class="{product.status}"><c:out value="{product.label}" /> by <c:out value="{product.brandLabel}" /> </h2>
<p><c:out value="{product.description}" /></p>

<h3>Ingredients</h3>
<ul>
<c:forEach items="{product.ingredients}" var="current">
<li>
<c:out value="{current.label}" />
<c:if test="{!empty current.allergens}">
```

product\_details.jsp

```
        <c:out value="/" /><c:forEach items="${current.allergens}" var="item" varStatus="status"><c:out value="${item.relation}" /><span class="${item.status}"> <c:out value="${item.label}" /></span><c:if test="${not status.last}"><c:out value=", " /></c:if></c:forEach><c:out value=")" />
    </c:if>
</li>
</c:forEach>
</ul>

<c:if test="${!empty product.allergens}">
<h3>Contamination issues</h3>
    <p>
        <c:forEach items="${product.allergens}" var="item" varStatus="status">This product <c:out value="${item.relation}" /> <span class="${item.status}"> <c:out value="${item.label}" /></span>
        </c:forEach>
    </p>
</c:if>
</div>
</c:if>

<c:if test="${empty product.label}">
<div class="uncertain" >
<p>Sorry, the product you scanned is not included in the data set. Please review it manually.</p>
</div>
</c:if>
<br />
<jsp:include page="available_actions.jsp" />
</body>
</html>
```

```
@CHARSET "UTF-8";

body{
font-family: Verdana, Helvetica, sans-serif;
font-size: 90%;
margin-left: 50px;
margin-right: 50px;
}
h1{
font-size: 1.2em;
}
h2{
font-size: 1.1em;
}
div.filter
{
padding: 20px;
border-style: solid;
border-width: medium;
border-color: #696969;
}
div.safe
{
padding: 20px;
border-style: solid;
border-width: medium;
border-color: #228B22;
}
h2.safe
{
color: #228B22;
}
div.uncertain
{
padding: 20px;
border-style: solid;
border-width: medium;
border-color: #FFD700;
}
span.uncertain
{
color: #FFD700;
font-weight: bold;
}
h2.uncertain
```

style.css

```
{
color:#FFD700;
}
div.unsafe
{
padding:20px;
border-style:solid;
border-width:medium;
border-color:#CD2626;
}
span.unsafe
{
color:#CD2626;
font-weight:bold;
}
h2.unsafe
{
color:#CD2626;
}
span.info
{
color:#696969;
}

div.debug
{
padding:20px;
border-style:solid;
border-width:medium;
border-color:#696969;
}
.button
{
border-style:solid;
border-width:medium;
border-color:#696969;
background:white;
padding:5px 10px;
color:black;
font-size:1em;
font-family:Verdana,Helvetica,sans-serif;
text-decoration:none;
vertical-align:middle;
}
.button:hover
```

style.css

```
{  
background: white;  
}  
.button:active  
{  
background: white;  
}
```

Ontology.owl.xml

```

<?xml version="1.0"?>
<!-- Abridged version -->
<rdf:RDF
  xmlns:skos="http://www.w3.org/2004/02/skos/core#"
  xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
  xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
  xmlns:assert="http://www.owl-ontologies.com/assert.owl#"
  xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
  xmlns:ao="http://www.inferenceTest.owl#"
  xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
  xmlns:rdfl="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:sqwrl="http://sqwrl.stanford.edu/ontologies/built-ins/3.4/sqwrl.owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:swrl="http://www.w3.org/2003/11/swrl#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:swrla="http://swrl.stanford.edu/ontologies/3.3/swrla.owl#"
xml:base="http://www.inferenceTest.owl">
  <owl:Ontology rdf:about="">
    <owl:imports rdf:resource="http://www.w3.org/2004/02/skos/core"/>
    <owl:imports rdf:resource="http://www.owl-ontologies.com/assert.owl"/>
    <owl:imports rdf:resource="http://swrl.stanford.edu/ontologies/3.3/swrla.owl"/>
    <owl:imports rdf:resource="http://sqwrl.stanford.edu/ontologies/built-ins/3.4/sqwrl.owl"/>
  </owl:Ontology>
  <!-- CLASSES -->
  <owl:Class rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
  <owl:Class rdf:ID="Brand"/>
  <owl:Class rdf:ID="Product"/>
  <owl:Class rdf:about="#Substance">
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2004/02/skos/core#Concept"/>
    <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Substance</rdfs:label>
  </owl:Class>
  <owl:Class rdf:ID="Ingredient">
    <rdfs:subClassOf rdf:resource="#Substance"/>
    <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Ingredient</rdfs:label>
  </owl:Class>
  <rdfs:Class rdf:ID="IngredientList">
    <rdfs:subClassOf rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#List"/>
    <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Ingredient list</rdfs:label>
  </rdfs:Class>
  <owl:Class rdf:ID="Allergen">
    <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Allergen</rdfs:label>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Allergens are displayed to the user for possible exclusion</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Substance"/>
  </owl:Class>

```

```

<!-- PROPERTIES -->
<owl:FunctionalProperty rdf:ID="hasIngredientList">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">hasIngredientList</rdfs:label>
  <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#List"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#TransitiveProperty"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:domain rdf:resource="#Product"/>
</owl:FunctionalProperty>
<owl:InverseFunctionalProperty rdf:about="#hasBrand">
  <rdfs:domain rdf:resource="#Product"/>
  <rdfs:range rdf:resource="#Brand"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:InverseFunctionalProperty>
<owl:FunctionalProperty rdf:ID="isBrandOf">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#InverseFunctionalProperty"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:domain rdf:resource="#Brand"/>
  <rdfs:range rdf:resource="#Product"/>
  <owl:inverseOf>
    <owl:InverseFunctionalProperty rdf:ID="hasBrand"/>
  </owl:inverseOf>
</owl:FunctionalProperty>
<owl:DatatypeProperty rdf:ID="hasProductDescription">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">has product description</rdfs:label>
  <rdfs:domain rdf:resource="#Product"/>
</owl:DatatypeProperty>
<owl:TransitiveProperty rdf:ID="contains">
  <rdfs:label xml:lang="en">contains</rdfs:label>
  <rdfs:label xml:lang="no">inneholder</rdfs:label>
  <rdfs:range rdf:resource="#Substance"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:TransitiveProperty>
<owl:TransitiveProperty rdf:about="#mayContain">
  <rdfs:label xml:lang="en">may contain</rdfs:label>
  <rdfs:label xml:lang="no">kan inneholde</rdfs:label>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:range rdf:resource="#Substance"/>
</owl:TransitiveProperty>
<owl:TransitiveProperty rdf:ID="maybeContaminatedBy">
  <rdfs:subPropertyOf>
    <owl:TransitiveProperty rdf:ID="mayContain"/>
  </rdfs:subPropertyOf>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:label xml:lang="en">may be contaminated by</rdfs:label>

```



```

<rdfs:label xml:lang="no">kan være kontaminert av</rdfs:label>
<rdfs:range rdf:resource="#Substance"/>
</owl:TransitiveProperty>
<owl:TransitiveProperty rdf:ID="mayContainTracesOf">
  <rdfs:label xml:lang="en">may contain traces of</rdfs:label>
  <rdfs:label xml:lang="no">kan inneholde spor av</rdfs:label>
  <rdfs:subPropertyOf>
    <owl:TransitiveProperty rdf:about="#mayContain"/>
  </rdfs:subPropertyOf>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:TransitiveProperty>
<owl:TransitiveProperty rdf:ID="derivedFrom">
  <rdfs:range rdf:resource="#Substance"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:subPropertyOf>
    <owl:TransitiveProperty rdf:about="#mayContain"/>
  </rdfs:subPropertyOf>
  <rdfs:label xml:lang="en">is derived from</rdfs:label>
  <rdfs:label xml:lang="no">er laget av</rdfs:label>
</owl:TransitiveProperty>
<owl:TransitiveProperty rdf:ID="producedAlongside">
  <rdfs:subPropertyOf rdf:resource="#mayContain"/>
  <rdfs:label xml:lang="en">is produced alongside</rdfs:label>
  <rdfs:label xml:lang="no">er produsert i en fabrikk som også produserer proudkter som inneholder</rdfs:label>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:TransitiveProperty>
<owl:TransitiveProperty rdf:ID="mayContainUpTo20ppm">
  <rdfs:subPropertyOf rdf:resource="#mayContain"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:label xml:lang="en">may contain up to 20 ppm</rdfs:label>
  <rdfs:label xml:lang="no">kan inneholde inntil 20 ppm</rdfs:label>
</owl:TransitiveProperty>
</rdf:RDF>

```