UNIVERSITY OF SOUTHAMPTON

Faculty of Engineering, Science and Mathematics
School of Electronics and Computer Science
Intelligence, Agents and Multimedia Group
Dependable Systems and Software Engineering Group

A mini-thesis progress report submitted for transfer from MPhil to PhD

Supervisor: Hugh Glaser
Supervisor: Dr Les Carr
Examiner: Dr Nicholas Gibbins

On the practical modeling of conceptual overlap among multiple facets in ontology domain concepts

by Benedicto Rodriguez-Castro

December 14th, 2007

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

Faculty of Engineering, Science and Mathematics
School of Electronics and Computer Science
Intelligence, Agents and Multimedia Group
Dependable Systems and Software Engineering Group

A mini-thesis progress report submitted for transfer from MPhil to PhD

by Benedicto Rodriguez-Castro

This report presents a study on the practical modelling of the conceptual overlap that might exist among the multiple facets that define a particular ontology domain concept. The notions of conceptual overlap and facet are defined, together with their relation to scenarios of multiple inheritance in ontology models. Starting from the notion of a *value partition*, a terminology of ontology modelling constructs is introduced that allows the characterization of two types of conceptual overlap with respect to the domain concept being examined: internal and external. These considerations make *explicit* some of the *implicit* modelling decisions taken previously in the field of ontology modelling. It also puts forward, a methodology to address this problem in a structured manner comprised of a series of steps which include a specific entry and exit criterion. The contribution of this research is proven with the modelling of the conceptual overlap in the "Fault" ontology domain concept that is part of the ReSIST project.

Table of Contents

List of Figures

Thesis

This work investigates the development of a set of measurable and practical guidelines for ontology modelling using the OWL language to represent conceptual overlap among multiple facets in domain concepts in the context of the Semantic Web.

This set of measurable and practical guidelines would allow ontologists to develop an ontology model that would outperform comparable candidates for a particular application driven parameter according to a relevant evaluation framework to be applied.

To test this hypothesis, different practical approaches to model a commonly agreed domain prone to a high degree of conceptual overlap will be proposed and measured against an evaluation framework that will show advantages and disadvantages of every model when compared with each other.

1. Introduction

The original idea at the beginning of this PhD program was to evaluate and understand the current state of Knowledge Technologies, specifically ontologies and Ontology Engineering and assess if these technologies could be applied to the field of Software Engineering to assist development teams in improving the quality of the artefacts delivered along the software development process. However, the evaluation of the current state of Ontology Engineering, uncovered new areas of research interest linked to some of the difficulties encountered when following any of the methodologies available to create ontologies.

Ontologies have emerged as one of the key components needed for the realization of the Semantic Web vision [Berners-Lee et al., 2001] and they bring with them a broad range of development activities that can be grouped into what is called Ontology Engineering. A detailed overview of what is an ontology, including the evolution of its definition in the literature, can be found in section 1.2 of [Gomez-Perez et al., 2004].

Ontology Engineering practices present many similarities to those in the Software Engineering field and there have been different adaptations of software engineering principles to the ontology engineering domain [Fernandez-Lopez et al., 1997].

Below is a list of the most common ontology engineering practices and a brief description of the work that each one of them entails [Gomez-Perez et al., 2004] [Fernandez-Lopez, 2002] [Fernandez-Lopez et al., 1997]. This list is not intended to be exhaustive given that new ontology engineering activities continue to appear as ontologies and the applications they are used for, keep on evolving.

- Requirements specification. Similarly to its Software Engineering counterpart, the main deliverable of this activity is an ontology requirements document.

- Conceptualization. This activity produces a conceptual model of the ontology, starting from a glossary of terms that contains the relevant domain knowledge for the ontology.

- Implementation. It constitutes the actual coding of the ontology into a formal ontology language that is machine-readable, such as the Web Ontology Language (OWL), [Dean and Schreiber, 2004].

- Evaluation. This activity could be seen as the Verification and Validation tasks performed in the Software Engineering discipline. The idea is to corroborate that the delivered ontology meets the requirements it was built for.

- Documentation. It is an important task that takes place throughout the ontological engineering process in order to understand the built ontology and enable potential future re-use. However, lack of guidelines on how to generate this documentation has been a challenge for ontologists when undertaking this activity, [Skuce and Lethbridge, 1995].

- Evolution and maintenance. This practice deals with the repercussions of modifications made to a deployed ontology in the applications and systems that the ontology operates. Management of change.

- Modularization. It studies how the construction of large ontologies can be realized by combining self-contained, independent and reusable knowledge components [D'Aquin, Schlicht et al., 2007].

- Extension. In situations when an ontology is re-used, it may be necessary to add new classes, properties, or other functionality to adapt it to new requirements. The process of adding or expanding the capabilities of an ontology is also referred to as ontology extension.

- Specialization or refinement. It could be viewed as the contrary process to ontology extension. In this case, the ontology functionality that is not relevant to meet its requirements is subtracted.

- Pruning or winnowing. It is characterized by tailoring, simplifying, or shrinking an ontology with respect to the needs of the application that is using it [Ehrig et al., 2004] [Alani et al., 2006].

- Integration. It deals with the question of how and whether to use all or part of ontologies that already exist [Uschold et al., 1996].

- Merging. It examines similarities and differences between source ontologies and it aims to produce a single ontology resulting from the combination of all the sources [Noy and Musen, 2000].

- Mapping or alignment. Like in the case of ontology merging, ontology mapping also involves looking at links between existing ontologies to make them consistent with one another, although here, the sources involved will be kept separately [Noy and Musen, 2000].

- Reasoning. This activity deals with the study of the inferring capabilities of the produced ontology.

Out of all these aspects of ontology engineering, this report primarily focuses in the ontology conceptualization task described above, and on the opportunities for improvement in the current state of the art methodologies.

The first part of the conceptualization consists on developing a glossary of terms representative of the target domain, obtained during the preceding knowledge acquisition phase. At this point, the construction of the model for the ontology starts and it is at this point that ontologists will have to solve different modelling issues to convert the glossary of terms into an ontology model. For example, what terms in the glossary should be modelled as classes? What terms should become properties, property values, or instances? This is the specific step in the conceptualization process that this research is intended to focus on [Noy and McGuinness, 2001].

Another level of complexity that will be considered, involves domain concepts that present a high degree of conceptual overlap among multiple facets used to define them. Ontology creation methodologies provide some guidelines on how to approach this design step however they do not seem to provide enough level of detail. This deficiency could lead ontology designers into making incorrect modelling choices relying on a subjective interpretation of the problem. The aim of this research is to explore this aspect of the ontology creation in depth and try to propose a set of practical and measurable guidelines that could assist ontologists in solving this issue in a more deterministic, reproducible and objective manner.

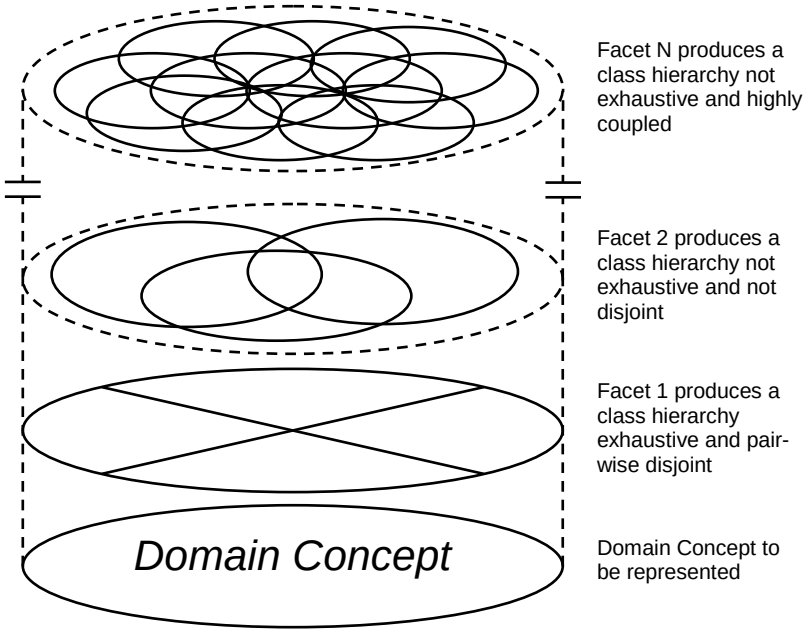

Figure 1. Representation of internal conceptual overlap.

Conceptual overlap refers to the intersection that exists among the considered definitions of a concept. Two types of conceptual overlap are identified: internal and external conceptual overlap.

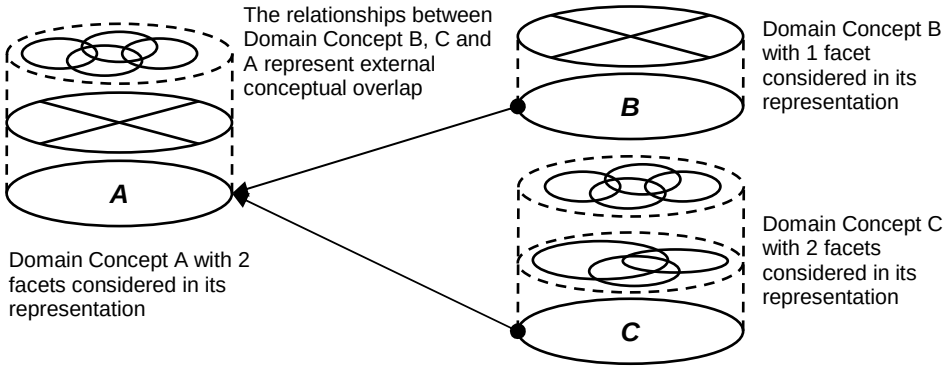

Figure 2. Representation of external conceptual overlap

Internal conceptual overlap refers to the intersection among the definitions considered within the same ontology domain concept. Each definition is normally linked to a facet that exists of the concept, (Figure 1).

External conceptual overlap refers to the intersection that could occur among two or more distinct ontology domain concepts due to the existence of certain relationships among them, (Figure 2).

A facet represents a criterion that would render a hierarchical taxonomy of the concept. When multiple facets are taken into account, the result is a poly hierarchical taxonomy of that concept. This notion of facet extends the definition of "value partition" introduced in [Rector, 2005].

When there is overlap among the facets, the poly hierarchical taxonomy will exhibit scenarios where certain terms will overlap each other across the taxonomies involved. Multiple inheritance provides a view of such scenario. The expression multiple inheritance in this context, refers to the situation where a term in the developed ontology is subsumed by two or more different terms in the ontology. In the case of ontology development using the OWL language this situation could apply to both OWL Classes and OWL Properties.

It is important to note the distinction between "conceptual overlap" and "multiple inheritance". Throughout this report conceptual overlap is considered an ontology design problem while multiple inheritance is viewed as one of the possible approaches to address and illustrate this problem, but not the only one.

To obtain a better idea of the multiple inheritance landscape for the ontologies in the Web, Figure 2 in [Wang et al., 2006] shows the shape of class hierarchies for the 1275 ontology files in the survey, (688 OWL and 587 RDFS ontologies).

Out of the 688 OWL ontologies, 122 were Directed Acyclic Graphs (DAGs), (17.7%) and 64 were multitrees (9.3%). This gives a total of 27% were most likely some type of conceptual overlap modelling in their class hierarchy is taking place. In the inferred ontology this number goes up to 30.2%.

In the case of RDFS ontologies, out of 587 included in the survey, a total of 77 (13%) had a DAG (6.8%) or a multitree (6.3%) as the shape of their class hierarchy.

The combined result is that about 20% of all ontologies on the Web (considered in the survey) include some type of multiple inheritance modelling scenario. This value seems too low based on how common multiple inheritance occurs in the real world. A possible interpretation for this could be due to a lack of best practice guidelines on how to model this problem, which in turn could be causing ontology developers to find creative ways to circumvent it.

On a similar study [d'Aquin, Baldassarre et al., 2007], surveys indicate that the number of ontologies and their presence in the traditional Web increases rapidly according to the latest figures. The number of OWL and RDF-S ontologies available online is approximately 6200 and 1700 respectively. These numbers are in the order of nearly ten times larger in the case of OWL ontologies and more than double for RDF-S when compared to the survey in [Wang et al., 2006] about a year earlier. The latter reported 688 and 587 OWL and RDF-S ontologies respectively.

This seems to indicate that since the adoption of the OWL specification language as a W3C standard in 2004 [Dean and Schreiber 2004], the ontology development community has been active and embraced the latest technology available in a detriment to its RDF-S predecessor. More importantly, it brings an interesting question to the forefront. How are these ontologies being built? What modelling problems and challenges are ontology developers facing and what approaches are they taking to solve them?

The rest of this report is organized as follows: Section 2 presents an overview of the main research areas in connection to the problem being discussed here. Section 3 describes the methods employed and the rationale behind them to address the problem. Section 4 outlines the conclusions gathered from this effort and Section 5 does the same regarding the lines open for further investigation.

1.1. The ReSIST Project

The contribution of this research is put into practice with the modelling of the conceptual overlap in the "Fault" ontology domain concept that is part of the ReSIST[1] project. ReSIST stands for Resilience and Survivability in Information Society Technology (IST) and it is a Network of Excellence (NoE) project funded under the Sixth Framework Programme of the European Union (ReSIST, 2006).

One of the objectives of the ReSIST project is to create a Knowledge Base (KB) application in the domain of resilient computing, partly inspired by the features demonstrated by the semantic web application CS AKTive Space [Glaser et al., 2004] [Shadbolt et al., 2004] and with many of the same requirements.

The aim of the ReSIST Knowledge Base (RKB) is to provide an application to the end-user that could serve as a portal to browse and search all type of information in the field of resilient computing: projects, people, institutions, publications, communities of practice, courses, etc. Meeting those requirements requires the development of an ontology fit for purpose in the domain of resilience and survivability in computer systems that will have to be built from scratch.

Further information of the main components and technologies being used to develop the ReSIST KB application can be found in [Glaser et al., 2007] [Anderson et al., 2007] [Millard et al., 2006].

[1] http://www.resist-noe.org/

2. Related Research

Reiterating from the previous section, the problem of modelling conceptual overlap among multiple facets of ontology domain concepts can be seen as a specific problem scenario within the broader activity of ontology conceptualization. Its solution involves several areas of research among which, four of them stands out. These are: ontology modelling, more specifically ontology design patterns, multiple inheritance, and ontology evaluation. This section outlines the latest developments and role of these four topics in the target of this research.

2.1. Ontology Modelling

There are several methodologies and approaches to build ontologies from scratch that address the topic of ontology conceptualization and more specifically ontology modelling. A comprehensive survey of the most relevant is provided in [Fernandez-Lopez et al., 2002]. However, these methodologies do not provide enough information at the specific level of detail. They look at ontology conceptualization and modelling in broader terms, from a higher level perspective, or from the point of view of what role in the overall ontology engineering lifecycle it plays and what dependencies it has with other engineering activities. Different methodologies provide different levels of detail on how ontology conceptualization can be performed, but none of them discuss in depth the modelling problem subject of this research (or its possible solutions) [Gomez-Perez et al., 2004] [Uschold and King, 1995] [Gruninger and Fox, 1995] [Sure and Studer, 2002]. Additionally, the methodologies referenced above are dated prior to the adoption of OWL by the W3C as the preferred ontology modelling language for the Semantic Web, and therefore, modelling elements specific to OWL are not taken into account. This is an important shortcoming, given that this research intends to solve the issue of conceptual modelling in the context of OWL.

There is however, an example of previous work that examines ontology modelling issues at the level of detail required for this research. This is [Noy and McGuinness, 2001], which in turn, bases part of its rationale on the principles of object-oriented modelling [Rumbaugh et al., 1991]. In this case, a step by step methodology is put forward to develop an ontology. Each step details the most relevant modelling decisions to be made. In their considerations they point out that there is not a single correct way or methodology for developing ontologies. It is always and necessarily an iterative process and the best approach differ depending on the application that one has in mind.

An alternative approach to ontology building is presented in [Good et al., 2006]. The authors opted for a team of volunteers untrained in the principles of knowledge engineering to develop a specific ontology in the medical science domain. Volunteers are guided by protocol to create and developed the ontology consisting of a web-based interface.

[Prieto-Diaz, 2003] shows an interesting approach at building an ontology using principles of faceted classification. However, this method requires the use of a postulated ontology which is not built using the method itself. This is not a trivial prerequisite and hence it only addresses our modeling needs partially.

In summary, there are several ontology construction methodologies available in the literature, however except for [Noy and McGuinness, 2001], they do not provide enough detailed information about the ontology conceptualization and implementation phase. None of them treat the activity of ontology conceptualization in the context of the OWL Semantic Web modeling language and none of them addresses the specific problem of modeling conceptual overlap in the terms described in this report.

2.1.1. Ontology Design Patterns (ODPs)

Within the area of ontology modelling there is an activity that is receiving a significant amount of attention, possibly due to the preceding success achieved in the field of software engineering. This activity is design patterns [Gamma et al., 1995]. A design pattern would be the ideal artefact for the modelling guideline this research would like to propose to address the specific problem of conceptual overlap under study. Related work in ontology design pattern that has been considered includes [Blomqvist and Sandkuhl, 2005] [Suarez-Figueroa et al., 2007] [Gangemi, 2005] together with the documents released as part of the World Wide Web Consortium (W3C) Semantic Web Best Practices and Deployment Working Group (SWBPD-WG)[2] [Noy, 2004] [Rector, 2005] [Noy and Rector, 2006] [Rector and Welty, 2005].

[Blomqvist and Sandkuhl, 2005] proposes a classification scheme for ontology patterns in Ontology Engineering composed of five levels, which are from top to bottom: Application Patterns, Architecture Patterns, Design Patterns, Semantic Patterns, Syntactic Patterns. It also provides a brief review on the status of maturity and adoption of each one of them in the ontology development field.

[Suarez-Figueroa et al., 2007] also talks of ontology patterns at different levels in this case in the context of networked ontologies. It distinguishes three: Logical ODPs, Architectural ODPs and Content ODPs. In broad terms, Logical ODPs are equivalent to the modeling elements provided by OWL or to compositions of them. Architectural ODPs are equivalent to Logical ODPs or composition of them and characterize the structure of the ontology determining "how an ontology should look like". A basic example of Architectural ODPs would be a "taxonomy". Lastly, Content ODPs are made of Logical ODPs instances or composition of them and attempt to solve a specific domain modeling problem. The *Participation*, *Role-Task*, *Design-Artifact* ODPs introduced in [Gangemi, 2005] can be seen as examples of Content ODPs.

Based on these two classification schemes of ontology patterns, the solution to our modeling problem, on one hand would include elements from the Design, Semantic and Syntactic Patterns described in [Blomqvist and Sandkuhl, 2005] and on the other, would fall into the Content Design Patterns as presented in [Suarez-Figueroa et al., 2007].

Special mention deserve, the documents released by the Semantic Web Best Practices and Deployment Working Group of the W3C. They provide an analysis of different ontology modeling problems at the precise level of detail intended by this research and they are discussed in the context of OWL as the implementation tool. Particularly [Noy, 2004] and [Rector, 2005], given that their common approach to represent

[2] http://www.w3.org/2001/sw/BestPractices/

property values as anonymous individuals will become a central piece to the conclusions claimed by our work which are discussed in the following section.

In summary, to the best of our knowledge, the area of ontology design patterns does not address the issue of conceptual overlap or its implications when it occurs among multiple facets of an ontology domain concept. There is no evaluation of different plausible approaches to tackle the problem either.

2.2. Multiple Inheritance (MI)

Another topic of research involved in the modelling of conceptual overlap is multiple inheritance. Multiple inheritance is often the most common manifestation of conceptual overlap and it has a theme of extensive research in the field of object-oriented design and programming. However, there are crucial differences between the field of object-oriented application design and ontology construction that condition to what extent the findings in the object-oriented paradigm can be extrapolated to the ontology modelling world. A good discussion regarding the differences between the two disciplines takes place in [Knublauch et al., 2006] which covers ontology development from an object-oriented developer point of view.

A very informative analysis regarding the need for MI in object-oriented languages, and in the C++ language particularly, takes place in [Cargill 1991] and [Waldo 1991]. Cargill claims no need for MI based on the lack of an example that will prove the need for it and it provides comprehensive mechanisms that do not require MI to achieve the same functionality. Waldo on the other hand, identifies three different types of MI: *implementation*, *interface* and *data*. According to this distinction, Cargill is solely referring to *implementation* MI. At the same time, Waldo provides a compelling example of *interface* and *data* MI that cannot be addressed by Cargill alternatives which sustains the need for the feature in the C++ language.

Unlike in the case of C++, the Java object-oriented language opted for not allowing multiple inheritance across classes. In Java, a class can only inherit behaviour and implementation from a single *parent* class. However, Java introduces the concept of *interface* conformance. Java interfaces could be seen as abstract classes, (where no implementation is provided). Java allows classes to implement or conform to multiple interface classes, which in turn can provide certain support for the type of multiple inheritance labelled by [Waldo, 1991] as *interface* MI. [Tempero and Biddle, 2000] provides an overview of different implementation techniques to simulate MI in the Java language and the limitations that still exists. The MI simulation is achieved by combining *single inheritance*, *delegation* and *interface conformance*.

As stated in the introductory section, this report regards multiple inheritance as one of the possible implementations in which the problem of modelling conceptual overlap in ontology domain concepts can be represented, but not the only one. In other words, an ontology exhibiting multiple inheritance among its terms implies existence of conceptual overlap, while conceptual overlap doesn't necessarily implies that multiple inheritance has to exist in the resulting ontology.

2.3. Ontology Evaluation

Lastly, ontology evaluation is also needed in order to have a framework where the performance of the proposed ontologies to solve the problem can be measured. It is

important to note however, that ontology evaluation, a broad research area in itself, is not a research objective in this report. Instead it is used as a supporting tool to allow the analysis and comparison of the proposed ontology modelling options.

There are several approaches in the field of ontology evaluation such as: application usage, decision criteria definition, use of a gold standard, data-driven and logical consistency. The rationale behind these approaches is outside the scope of this report and the reader is referred to [Vrandecic, 2006] which provides a concise overview of the methodologies, together with the most relevant works within each one of them.

The initial framework for ontology evaluation used in this report is derived from the documents released by the W3C Semantic Web Best Practices and Deployment Working Group. These documents address evaluation of ontology modelling decisions at the content design level using the terminology in [Suarez-Figueroa et al., 2007], which fits the purpose of this research. The framework and the rationale behind it, is covered in detail in the next section.

3. Methods

This section introduces a methodology to undertake the problem of modelling conceptual overlap in ontology domain concepts in a structured manner.

The methodology comprises several steps which are outlined below:

Step 1. Define domain concept to be modelled
Step 2. Develop different ontology models to represent the domain concept
Step 3. Populate same set of individuals in all models
Step 4. Develop a test suite of user questions for all models
Step 5. Select an evaluation framework for all models
Step 6. Analyze results for all models

Essentially, different ontology models are proposed to represent the ontology domain concept and an evaluation framework is selected to determine the model that optimizes the desired evaluation parameter(s).

Every step identifies the entry criteria to proceed with the activities involved in such step and the exit criteria to progress onto the next one. Consequently, the entry criteria of any given step, matches the exit criteria of its predecessor [CMMI Product Team, 2006].

Step 1. Define the domain concept

In this step, the ontology modeller has to identify the domain concept subject of the model and the different facets that will be considered.

Every facet produces a hierarchical classification. Because of the presence of conceptual overlap among the facets, certain terms would appear in multiple nodes of the classification hierarchies obtained.

This elicitation process results in a closed vocabulary of all the relevant terms involved in the representation of the domain concept.

The classification hierarchies and the closed vocabulary of terms determined by them are the required entry criteria for the different ontology models to be proposed in step 2.

In the case of ReSIST, the target domain concept to be represented is the concept of Fault as defined in [Avizienis et al., 2005]. Figures 3, 4 and 5 illustrate the different classification of the concept of “Fault” that should be captured in the ontology for ReSIST. The background rationale of the figures in the context of dependable and secure computing can be further studied in [Avizienis et al., 2005].

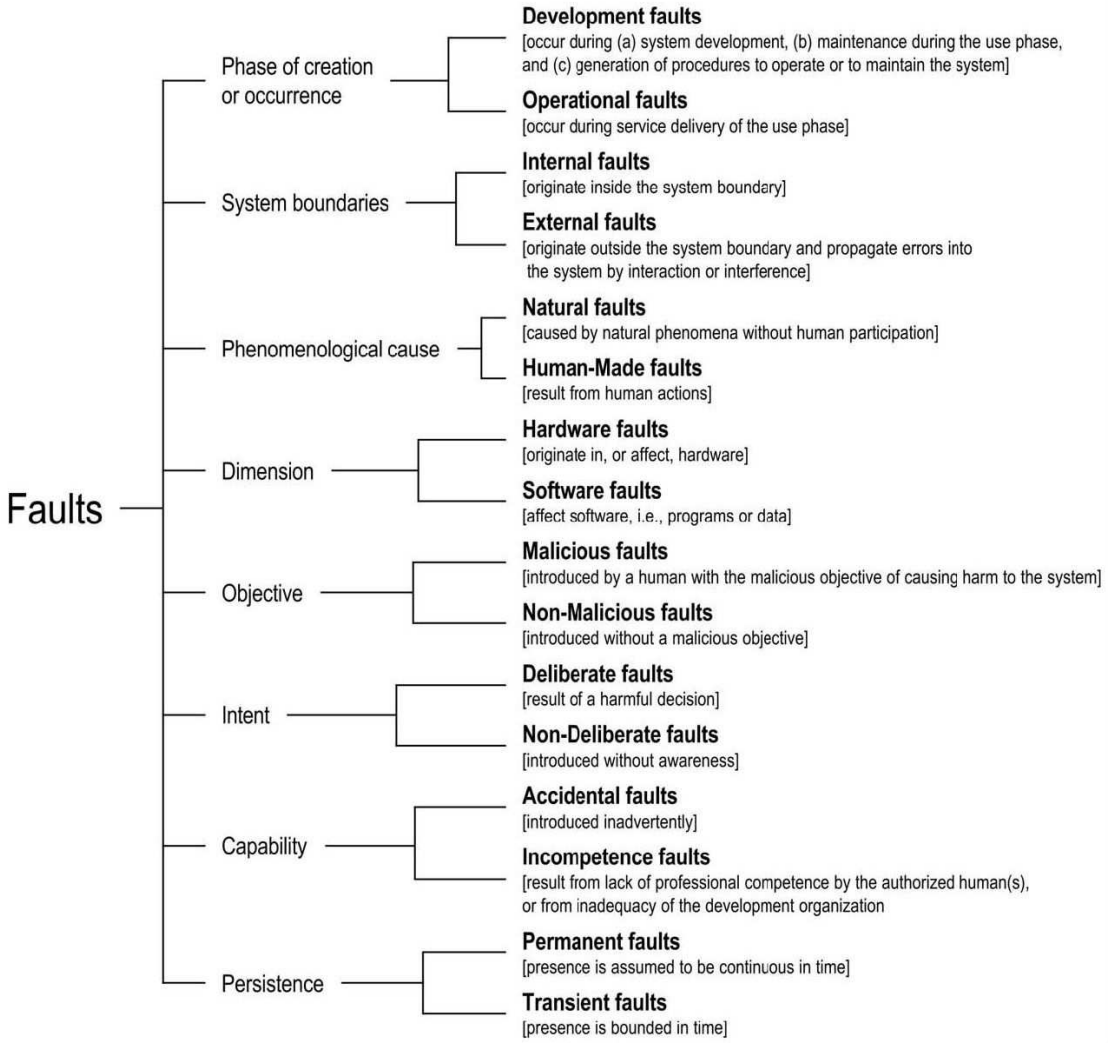

Figure 3.The elementary fault classes [Avizienis et al., 2005].

Figure 3 shows the first level of the tree diagram, which is referred to as the eight basic viewpoints. The eight basic viewpoints lead to the elementary fault classes shown in the second level of the tree.

The cited publication also notes that if all combinations of these 8 elementary classes were possible, the total number of combined fault classes would be 256. However not all combination occur in reality and Figure 4 and 5 illustrate the 31 most likely combined fault classes as a tree and a matrix representation respectively.

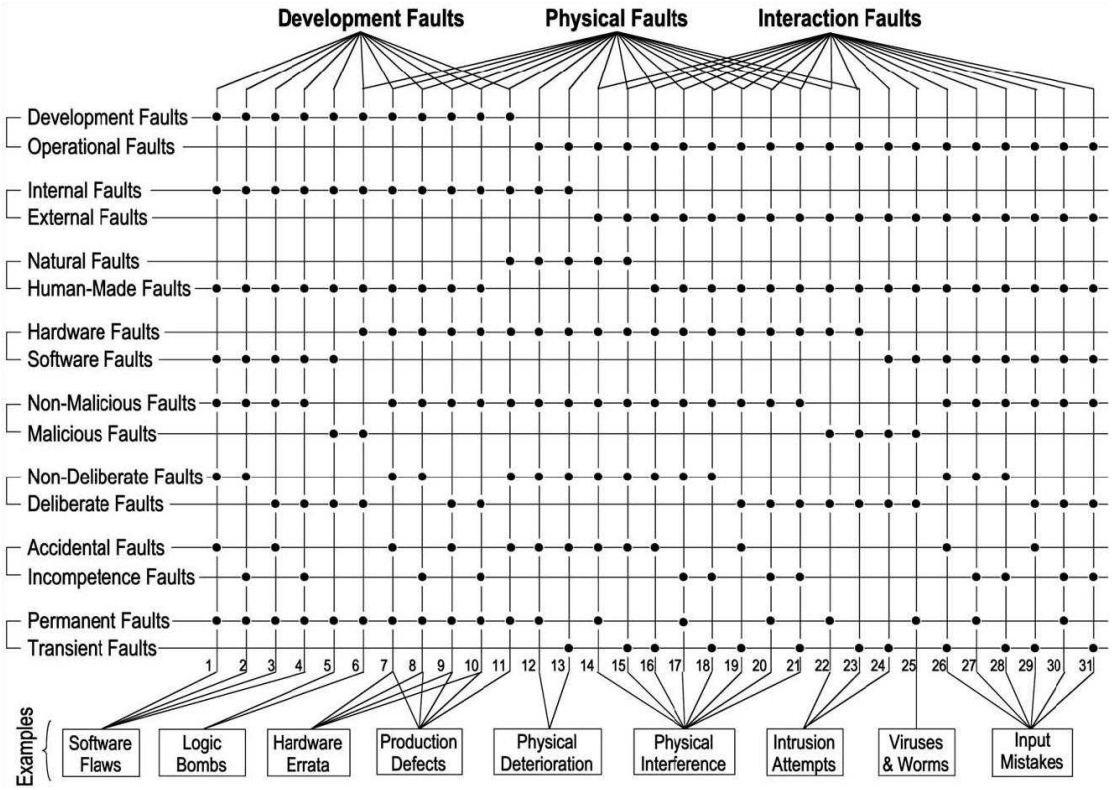

Figure 4. Matrix representation of the classes of combined faults [Avizienis et al., 2005].

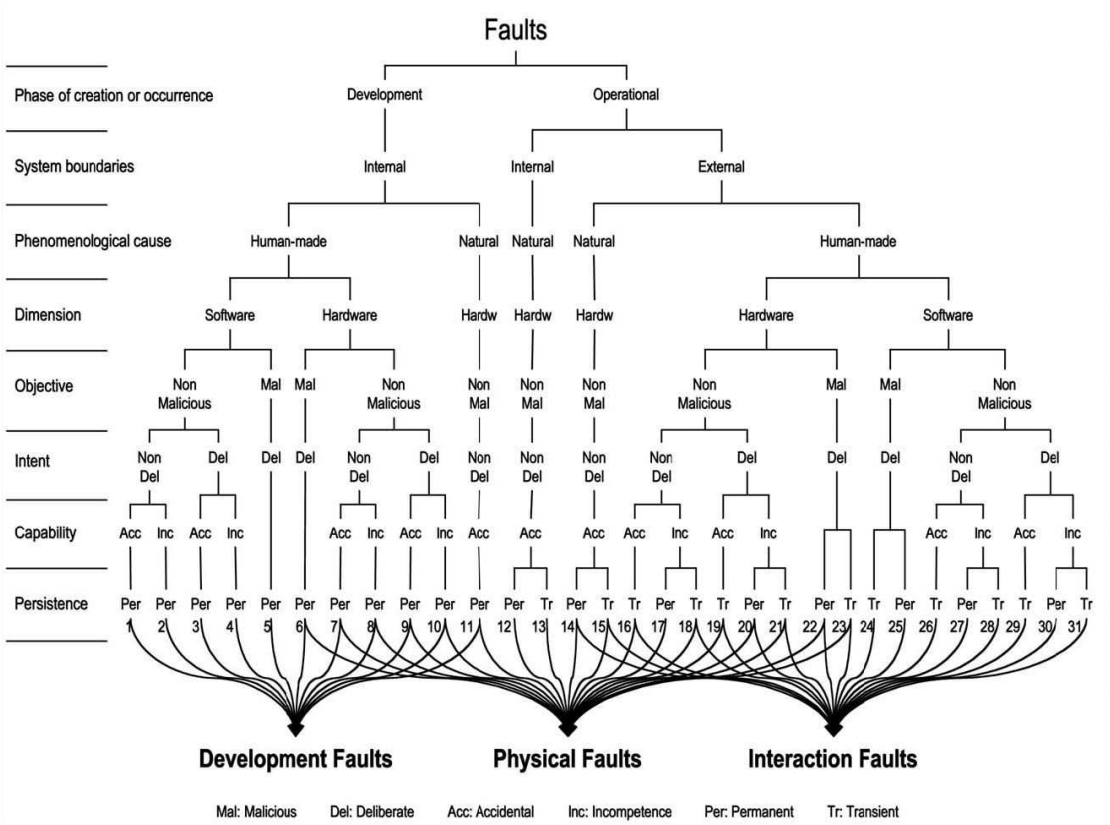

Figure 5. Tree representation of the classes of combined faults [Avizienis et al., 2005].

From figures 3, 4 and 5, four different facets of the Fault domain concept are identified:

- The 8 elementary fault classes
- The 3 major partially overlapping groupings
- The 9 illustrative examples of fault classes
- The 31 likely combined fault classes out of 256 possible combined fault classes

Each facet renders a different hierarchical classification that can be seen as four different classification hierarchies superimposed of the domain concept of Fault. The four classification hierarchies are outlined below:

Classification 1:

- Faults eight basic view points
    - Phase of creation or occurrence
        - Development faults
            - Fault type 1, …, Fault type 11
        - Operational faults
            - Fault type 12, …, Fault type 31
    - System boundaries
        - Internal faults
            - Fault type 1, …, Fault type 13
        - External faults
            - Fault type 14, …, Fault type 31
    - Phenomenological cause
        - Natural faults
            - Fault type 11, …, Fault type 15
        - Human-made faults
            - Fault type 1, …, Fault type 10
            - Fault type 16, …, Fault type 31
    - Dimension
        - Hardware faults
            - Fault type 9, …, Fault type 23
        - Software faults
            - Fault type 1, …, Fault type 5
            - Fault type 24, …, Fault type 31
    - Objecive
        - Malicious faults
            - Fault type 5, Fault type 6
            - Fault type 22, …, Fault type25
        - Non-malicious faults
            - Fault type 1, …, Fault type 4
            - Fault type 7, …, Fault type 21
            - Fault type 26, …, Fault type 31
    - Intent
        - Deliberate faults
            - Fault type 3, …, Fault type 6
            - Fault type 9, Fault type 10
            - Fault type 19, …, Fault type 25
            - Fault type 29, …, Fault type 31

        - Non-deliberate faults
            - Fault type 1, Fault type 2
            - Fault type 7, Fault type 8
            - Fault type 11, …, Fault type 18
            - Fault type 26, …, Fault type 28
    - Capability
        - Accidental faults
            - Fault type 1, Fault type 3, Fault type 7, Fault type 9
            - Fault type 11, …, Fault type 16
            - Fault type 19, Fault type 26, Fault type 29
        - Incompetence faults
            - Fault type 2, Fault type 4, Fault type 8, Fault type 10
            - Fault type 17, Fault type 18
            - Fault type 20, Fault type 21
            - Fault type 27, Fault type 28
            - Fault type 30, Fault type 31
    - Persistence
        - Permanent faults
            - Fault type 1, …, 12
            - Fault type 14, 17, 20, 22, 25, 27, 30
        - Transient faults
            - Fault type 13, 15, 16, 18, 19, 21, 23, 24, 26, 28, 29, 31

Classification 2:

- Faults three major groups
    - Development faults
        - Fault type 1
        - …
        - Fault type 11
    - Physical faults
        - Fault type 6
        - …
        - Fault type 23
    - Interaction faults
        - Fault type 14
        - …
        - Fault type 31

Classification 3:

- Examples of nine known fault classes
    - Software flaws faults
        - Fault type 1, …, Fault type 4
    - Logic bombs faults
        - Fault type 5, Fault type 6
    - Hardware errata faults
        - Fault type 7, …, Fault type 10
    - Production defects faults

        - Fault type 7, …, Fault type 11
    - Physical deterioration faults
        - Fault type 12, …, Fault type 13
    - Physical interference faults
        - Fault type 14, …, Fault type 21
    - Intrusion attempts faults
        - Fault type 22, ..., Fault type 24
    - Viruses and worms faults
        - Fault type 25
    - Input mistakes faults
        - Fault type 26, …, Fault type 31

Classification 4:

- Thirty-one most likely combined faults
    - Fault type 1
    - Fault type 2
    - …
    - Fault type 31

The terms in the 4 classifications form a closed vocabulary. The classifications and the closed vocabulary are the required entry criteria to develop the ontology models of the Fault domain concept in the next step.

Step 2. Develop different ontology models to represent the domain concept

This step creates the proposed models that will be used in the evaluation. Each model represents a different approach to characterize the conceptual overlap existing in the target domain concept.

Some definitions will be reviewed and some terminology will be introduced that will be useful to understand the rationale and foundation behind the candidate models.

The starting point will be the documents released by the Semantic Web Best Practices and Deployment Working Group. Two of them in particular are of special significance: [Rector, 2005] and [Noy, 2004].

[Rector, 2005] introduces the concept of "value partition" as a modelling technique to represent features, attributes, or modifiers that describe other concepts in the ontology. In its definition, a class in the ontology is partitioned by a group of subclasses if:

a) The subclasses are mutually exclusive
b) The subclasses completely exhaust the parent class

For example: In Figure 6, (which corresponds to a version of Figure 4 in [Rector, 2005] with additional annotations), the class "HealthValue" is partitioned by the classes "Poor_health_value", "Medium_health_value" and "Good_health_value" according to the definition stated earlier. In other words, the three subclasses represent a "value partition" of the parent class.

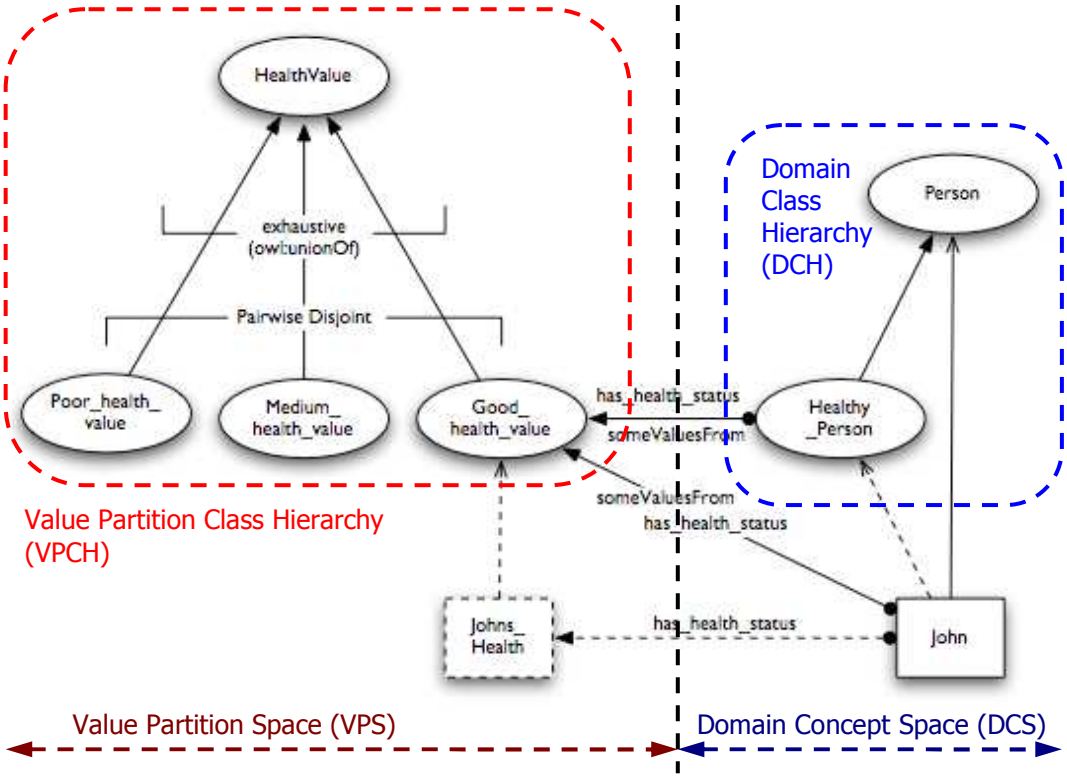

Figure 6. Extended version of Figure 4 in [Rector, 2005].
Using an anonymous individual as the value for the property "has_health_status".

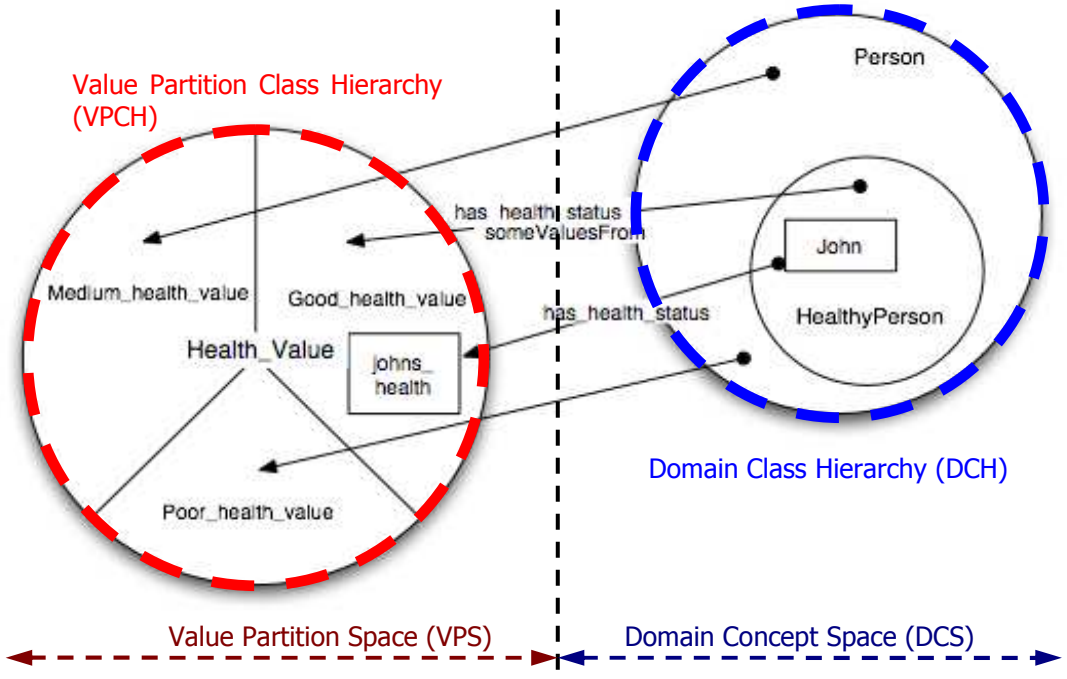

Figure 7. Extended version of Figure 3 in [Rector, 2005].
Venn-style diagram illustrating the concept of "value partition".

Alternatively, Figure 7 shows another view of the same scenario from Figure 6. Both figures depict the idea of using an anonymous individual that belongs to one of the

classes in the "value partition" as the value for a property in the ontology. This value is used to describe a specific attribute of the domain concept that participates in the property. See specifically Pattern 2, version 2 in [Rector, 2005].

[Noy, 2004], on the other hand, presents different modelling alternatives to the usage of a class as a property value. This situation may occur when a generic hierarchy of classes is used to annotate other classes or individuals in the ontology. Out of the five approaches presented in the document, it is particularly interesting approach 4 because it also employees anonymous individuals to qualify the value of a property for a domain concept in the ontology. However, the document does not place any restriction on the structure or properties that the generic class hierarchy may posses.

For example: In Figure 8 (which corresponds to a version of Figure 4 in [Noy, 2004] with additional annotations), the generic class hierarchy formed by "Animal", "Lion" and "AfricanLion" is used to provide the value for the subject of "books about animals", which is the domain concept the ontology intends to represent. Figure 9 is just a Venn-style diagram representation of the model in Figure 8.

To further discuss the repercussion that the parallelism between pattern 2, variant 2 in [Rector, 2005] and approach 4 in [Noy, 2005] in the representation of an anonymous individual as a property value, the following terminology is introduced:

Generic Class Hierarchy (GCH):

> The term Generic Class Hierarchy (GCH) refers to any hierarchy of classes with any hierarchy structure or shape among its classes and with or without conceptual overlap. For example: a single class or a set of classes organized in a list, a tree or a directed acyclic graph structure.

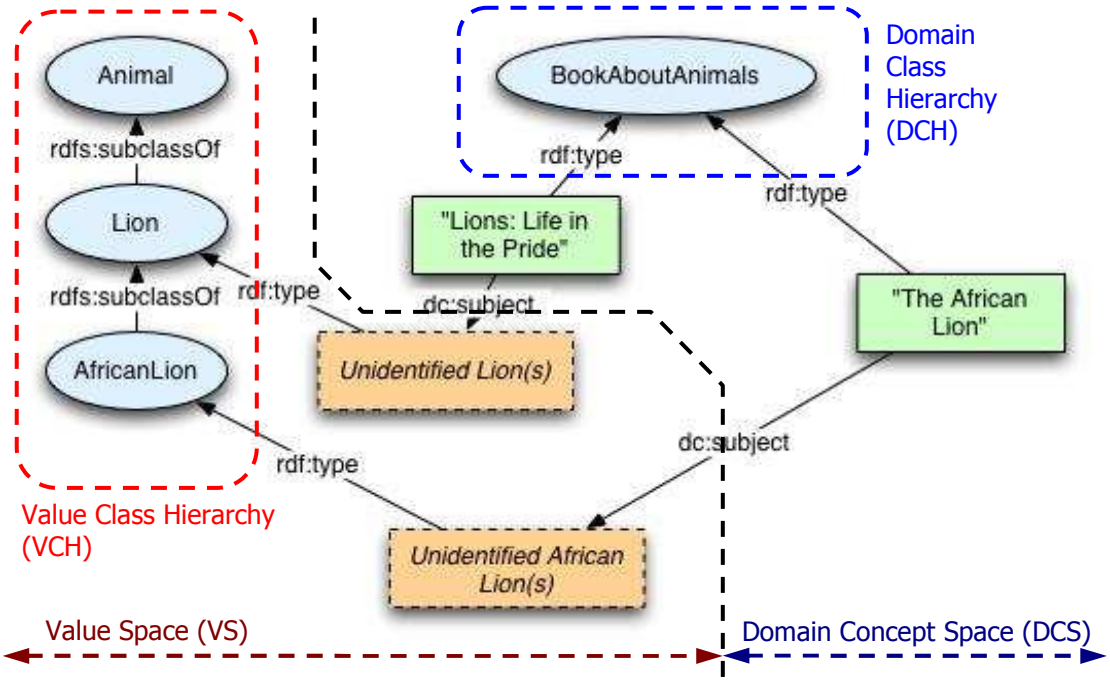

Figure 8. Extended version of Figure 4 in [Noy, 2004].
Using an anonymous individual as the value for the property "dc:subject".

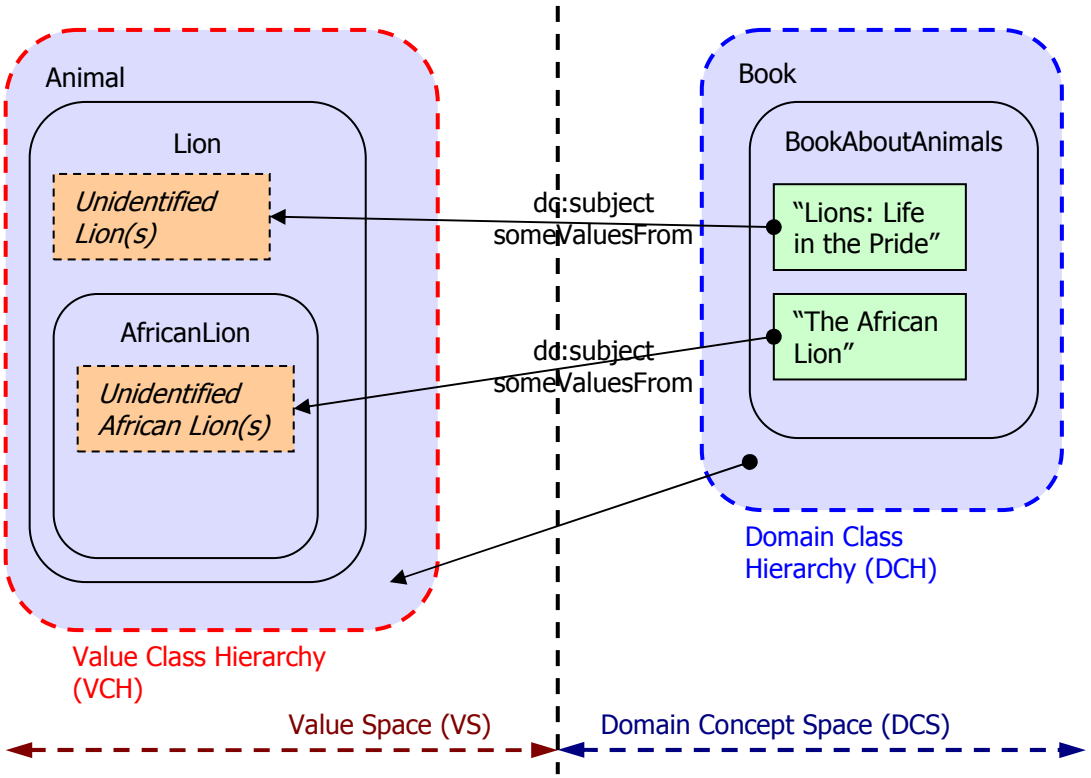

Figure 9. Venn-style diagram illustrating the concept of "value class hierarchy".

Domain Class Hierarchy (DCH):

The term Domain Class Hierarchy (DCH) refers to any Generic Class Hierarchy (GCH) that contains the classes corresponding to the domain concepts that the ontology is intended to represent. For example: in Figure 6 and 7 the DCH depicted is the GCH formed by the concepts "Person" and "Healthy Person" and in Figure 8 and 9 the DCH depicted is the GCH determined by the concept "Book About Animals".

Value Class Hierarchy (VCH):

The term Value Class Hierarchy (VCH) refers to any Generic Class Hierarchy (GCH) that is used to provide values to properties whose domains are the individuals of other classes in the ontology as defined in approach 4 of [Noy, 2004], (depicted in Figure 8 and 9).

Value Partition Class Hierarchy (VPCH):

The term Value Partition Class Hierarchy (VPCH) refers to the parent class and subclasses that are part of a "value partition" as defined in pattern 2 variant 2 of [Rector, 2005], (depicted in Figure 6 and 7).

Domain Concept Space (DCS):

The term Domain Concept Space (DCS) is the section of the ontology that contains the Domain Class Hierarchy(-ies) of the ontology. See Figure 6 and 8.

Value Space (VS):

The term Value Space (VS) is the section of the ontology that contains the Value Class Hierarchy(-ies) of the ontology. See Figure 8 and 9.

Value Partition Space (VPS):

The term Value Partition Space (VPS) is the section of the ontology that contains the Value Partition Class Hierarchy(-ies) of the ontology. See Figure 6 and 7.

From these definitions, it can be inferred that:

- A VPCH is a particular case of VCH, given that the former meets all the characteristics of the latter, plus the restrictions proper of a "value partition". However, the vice versa does not apply.

- Similarly, every VPS is also a VS but the reciprocal implication does not hold for the same reasons.

This characteristic is important because it allows the following conclusions:

Premise 1:

The Value Partition Class Hierarchy in Figure 6 plays the same role as the Value Class Hierarchy in Figure 8. This is, both class hierarchies provide the range for the property "has_health_status" and "dc:subject" in their ontologies respectively using an anonymous individual as the property value. (Even though, "has_health_status" is a functional property while "dc:subject" is not).

Conclusion 1:

A Value Class Hierarchy extends the "value partition" modelling pattern as described in [Rector, 2005] to any type of class hierarchy. This is, a Generic Class Hierarchy can also be used to act as a Value Partition Class Hierarchy for a property in the ontology. (Despite a Generic Class Hierarchy may not conform to the definition of "value partition").

Premise 2:

In ontologies that use value partitions, it is a good modelling practice to make disjoint the class hierarchies that represent the value partitions (VPCHs), from the class hierarchies that represent the domain concepts (DCHs), creating two distinct spaces in the ontology model [Horridge et al., 2004] [Rector, 2005]. This is, the Value Partition Space and the Domain Concept Space.

Conclusion 2:

(From Premise 1 and 2 it follows):
If a Generic Class Hierarchy (let's call it GCH1) is used to act as a Value Class Hierarchy, then it would be a good modelling practice to make GCH1 disjoint from the class hierarchies that represent the domain concepts in the ontology (DCHs).

For example: in Figure 8, the Value Class Hierarchy determined by the class "Animal" and its subclasses would be disjoint from the Domain Class Hierarchy

determined by the class "BookAboutAnimals", creating a disjoint Value Space and Domain Concept Space in the ontology.

Premise 3:

Say two ontologies O1 and O2, with two Domain Class Hierarchies DCH1 and DCH2 respectively.

Conclusion 3:

(From Premise 1, 2 and 3 it follows):
It is possible for a Domain Class Hierarchy (DCH1) in an ontology (O1) to act as a Value Class Hierarchy in another ontology (O2) and in that case:

a) DCH1 in O1 becomes the range for some property in O2, (using an anonymous individual from DCH1 as the property value) and also,
b) DCH1 in O1 becomes part of the Value Space in O2 and disjoint from DCH2 in O2.

For example: this could be the case in Figure 8, if the Value Class Hierarchy formed by “Animal” and its subclasses was imported from a separate ontology where it acted as a Domain Class Hierarchy.

Premise 4:

Say ontology O1, with two Domain Class Hierarchies DCH11 and DCH12.

Conclusion 4:

(From Premise 1 follows):
It is possible for a Domain Class Hierarchy (DCH11) to act as a Value Class Hierarchy for another Domain Class Hierarchy (DCH12) in the same ontology O1. In that case:

a) DCH11 in O1 becomes the range for some property in O1.
b) DCH12 in O1 becomes the domain for that same property in O1.
c) DCH11 in O1 causes the Value Space and the Domain Concept Space of O1 to overlap.

For example: this could be the case in Figure 8, if the Value Class Hierarchy was used to represent the domain concept of actual animals in addition to act as a “value partition” for the “BookAboutAnimals” class hierarchy via the property “dc:subject”.

These implications play an important role in the models proposed for the ReSIST concept of Fault because the Fault domain concept in ReSIST serves a twofold purpose, meeting the criteria in Conclusion 4, given that:

- It represents occurrences of real world faults in the field of resilient and dependable systems. In that sense, the model is a Domain Class Hierarchy (DCH) in the Domain Concept Space (DCS) of the overall ReSIST ontology.

- It provides the values for certain properties whose range is a type of fault. In that sense the model is a Value Class Hierarchy (VCH) in the Value Space (VS) of the overall ReSIST ontology. For example: the Fault domain concept will supply the value for properties such as hasResearchSubject (associated with the concept of Publication), hasResearchInterest (associated with the

concept of Person) or hasSupportFor (associated with the concept of Resilient Mechanism), whose range is a type of fault or faults in the Fault VCH. (See Figure 10).

This DCH and VCH duality reveals two scopes of conceptual overlap taking place in the overall ReSIST ontology with respect to the concept of Fault:

Internal Conceptual Overlap:

It is the conceptual overlap inside the Fault DCH among the multiple classification hierarchies of Fault. The scope of this conceptual overlap can be seen as "internal" to the Fault domain concept.

External Conceptual Overlap:

It is the conceptual overlap outside the Fault VCH with other DCH(s) in the ontology (such as: Person, Publication, or Resilient Mechanism). The scope of this conceptual overlap can be seen as "external" to the Fault domain concept.

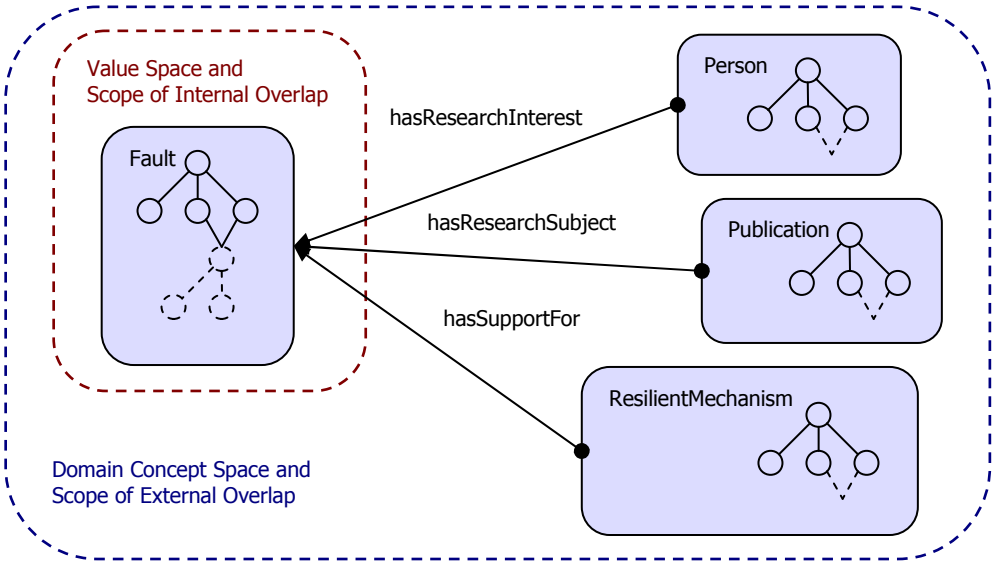

Figure 10. The Fault domain concept Value Class Hierarchy is part of the Domain Concept Space and the Value Space in the overall ReSIST ontology.

The rest of this section presents the ontology models proposed to represent the conceptual overlap in the Fault domain concept of ReSIST. The models are organized in two groups. The first group addresses the internal conceptual overlap in the Fault domain concept, and the second addresses the external conceptual overlap between the Fault domain concept and other domain concepts in the overall ReSIST ontology.

3.2.1. Modelling internal overlap

The required entry criteria for all the internal conceptual overlap models, are all the classification hierarchies and the closed vocabulary of terms determined by them, elicited in the previous step of this process.

3.2.1.1. Model 1. Using OWL Classes.

Model 1 is probably the most intuitive and straight-forward of all the models. The principal design criteria followed to create Model 1 are:

- Use the OWL Class to represent every term in the closed vocabulary produced in step 1.

- Use the class/subclass relation between OWL Classes to recreate the hierarchical structure of terms in all the classifications produced in step 1.

In our ongoing ReSIST example, the top class in the domain concept model is the term “Fault”. The direct subclasses of “Fault” are the terms that represent each one of the four facets that produce the classification hierarchies in step 1: “Faults eight basic view points”, “Faults three major groups”, “Examples of known fault classes” and “Thirty-one most likely combined faults”, which in the actual OWL implementation have been named as “BasicViewPointFault”, “MajorGroupFault”, “NamedClassFault” and “NamedCombinedFault” respectively. The process repeats for the rest of the terms in each one of the four classification hierarchies.

Note that terms that appear in multiple classification hierarchies due to the conceptual overlap in the domain concept will be subsumed by multiple OWL Classes. This is the case of the terms “Fault type 1”, “Fault type 2”, …, “Fault type 32”, ”, which in the actual OWL implementation have been named as the OWL Classes “FaultType1”, “FaultType2”, …, “FaultType32” respectively.

For example, the OWL Class named “FaultType10”, which represents the term “Fault type 10”, will be a direct subclass in the ontology model of the OWL Classes created from the following terms:

- From classification 1 in step 1:
  - “Development Fault”,
  - “Internal Fault”,
  - “Human-made Fault”,
  - “Hardware Fault”,
  - “Non-malicious Fault”,
  - “Deliberate Fault”,
  - “Incompetence Fault” and
  - “Permanent Fault”

- From classification 2 in step 1:
  - “Development Fault” and
  - “Physical Fault”

- From classification 3 in step 1:
  - “Hardware Errata Fault” and
  - “Production Defect Fault”

- From classification 4 in step 1:
  - “Thirty-one most likely combined faults”

Once all the terms and classification hierarchies have been represented, the set of OWL Classes in the ontology model will present the shape of a directed acyclic

graph. This model represents the conceptual overlap in the domain concept using multiple inheritance for the OWL Classes involved and can be in Figure 11.

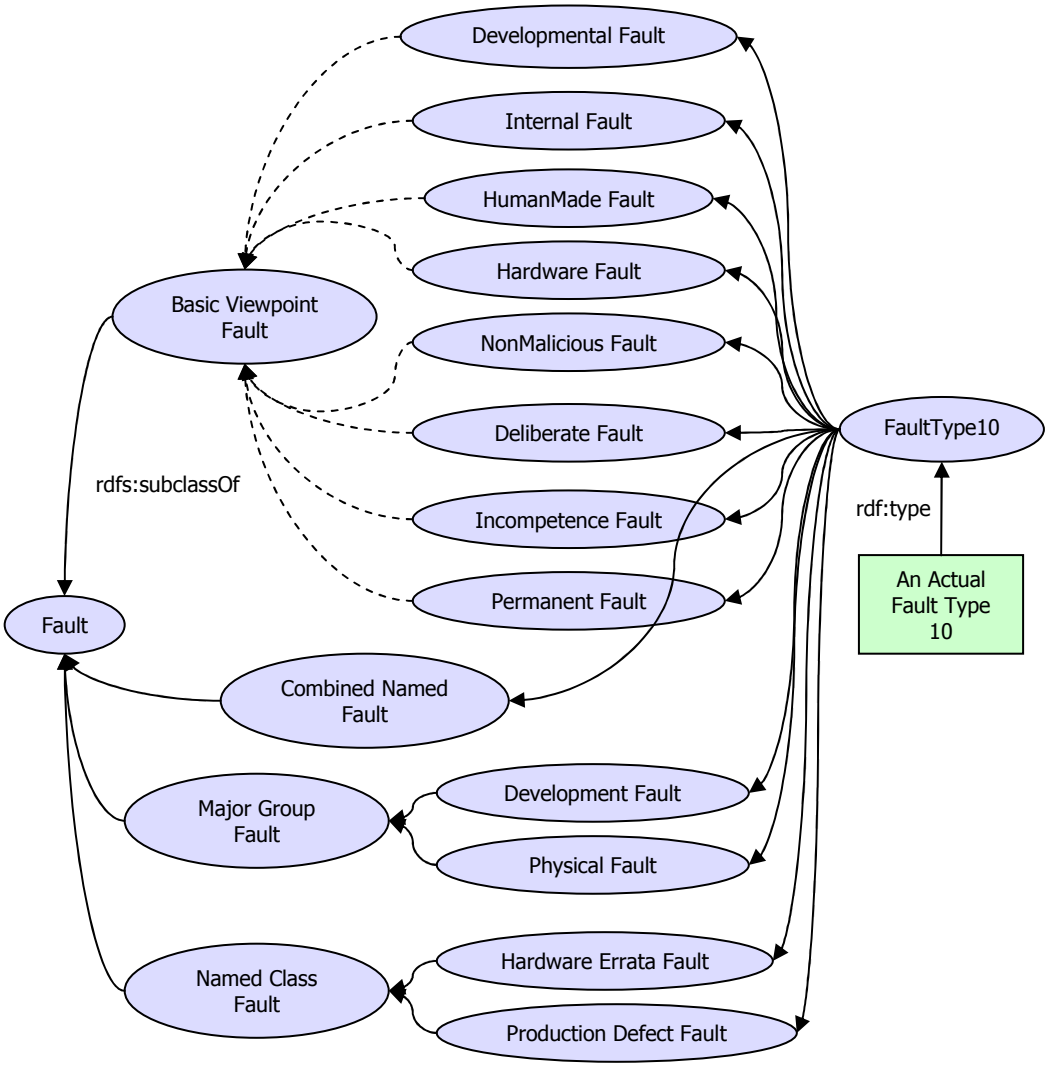

Figure 11. Example of Model 1.

3.2.1.2. Model 2. Using OWL Properties with subtype relations.

Model 2 can be seen analogous to Model 1 with a key variation: using the OWL Property instead of the OWL Class as the main modelling element. The principal design criteria followed to create Model 2 are:

- Use a single OWL Class to represent:
  - The entire "Fault" Domain Class Hierarchy (which in this case is composed of only one class),
  - The domain for all properties related to the "Fault" concept in the model and
  - The type for all "Fault" individuals.

- Use the OWL Property to represent every other term in the closed vocabulary produced in step 1 except for the domain concept itself "Fault".

- Use the property/subproperty relation between OWL Properties to recreate the hierarchical structure of terms in all the classifications produced in step 1.

This design approach is sustained by the fact that an OWL Property represents the anonymous class formed by all the individuals who have a relationship on that property [Horridge et al., 2004]. While in Model 1 individuals are characterized by the OWL Class(es) they belong to, in Model 2 they are characterized by the OWL Property(ies) they participate in.

In our ongoing ReSIST example, the top properties in the hierarchy are the ones that represent each one of the four facets that produce the classifications in step 1: "Faults eight basic view points", "Faults three major groups", "Examples of known fault classes" and "Thirty-one most likely combined faults", which in the actual OWL implementation have been named as "hasBasicViewPointFault", "hasMajorGroupFault", "hasNamedClassFault" and "hasNamedCombinedFault" respectively. The process repeats for the rest of the terms in each one of the four classification hierarchies.

Note that terms that appear in multiple classification hierarchies due to the conceptual overlap in the domain concept will be subsumed by multiple OWL Properties. This is the case of the terms "Fault type 1", "Fault type 2", …, "Fault type 32", which in the actual OWL implementation have been named as the OWL Properties "hasFaultType1", "hasFaultType2", …, "hasFaultType32" respectively.

For example, the OWL Property "hasFaultType10", which represents the term "Fault type 10", will be a direct subproperty in the ontology model of the OWL Properties created from the following terms:

- From classification 1 in step 1:
  - "Development Fault",
  - "Internal Fault",
  - "Human-made Fault",
  - "Hardware Fault",
  - "Non-malicious Fault",
  - "Deliberate Fault",
  - "Incompetence Fault" and
  - "Permanent Fault"

- From classification 2 in step 1:
  - "Development Fault" and
  - "Physical Fault"

- From classification 3 in step 1:
  - "Hardware Errata Fault" and
  - "Production Defect Fault"

- From classification 4 in step 1:
  - "Thirty-one most likely combined faults"

Once all the terms and classification hierarchies have been represented, the set of OWL Properties in the ontology model will present the shape of a directed acyclic graph as it happened in Model 1 with the set of OWL Classes. This model addresses the conceptual overlap in the domain concept using multiple inheritance for the OWL Properties involved.

To complete Model 2, two variations have been identified which are presented in the following sections.

3.2.1.2.1. Variation 1

These additional design characteristics provide Variation 1 to Model 2:

- Use the OWL Datatype Property to represent the terms.
- The range is the built-in XML Schema datatype xsd:boolean.

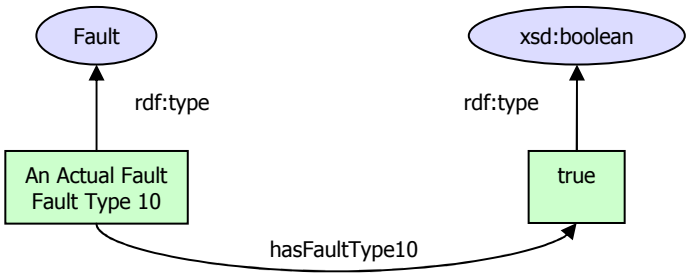

Figure 12. Example of Model 2 Variation 1.

3.2.1.2.2. Variation 2

These additional design characteristics provide Variation 2 to Model 2:

- Use the OWL Object Property to represent the terms.
- The range of all these properties is the OWL Class "Fault". This can be achieved using two different approaches:
    a) Using anonymous individuals as property values [Noy, 2005].
    b) Using the same individual as property domain and range value.

The use of an anonymous individual in option a) seems to be redundant given that the anonymous individual has an identical role to the individual that it is providing the value for. This reason favours option b) where the same individual is used as the domain and range for the property value.

In both variations, the transitive characteristic of the property/sub-property relation allows an individual that participate in the property to participate in its super-properties as well.

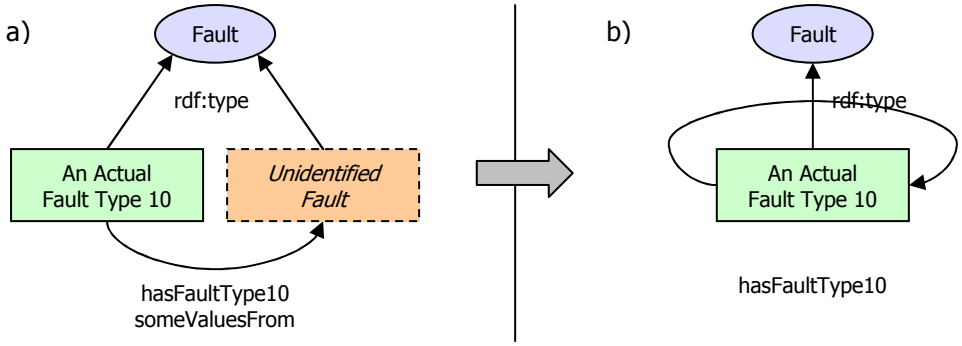

Figure 13. Example of Model 2 Variation 2.

Properties used as described in this model could be seen as *unary relations* for the individuals that participate in them. The property doesn't relate the individual with another individual in the ontology. Instead, it indicates that the property for that individual exists; it is true (Variation 1). Furthermore, in the case of (Variation 2b) properties could be seen as *reflexive* given that they relate the individual to itself.

3.2.1.3. Model 3. Using OWL Properties without subtype relations.

Model 3 is identical to Model 2 with a key variation: the hierarchical structure of terms in the classifications from step 1 is not represented in the set of OWL Properties. Therefore, the principal design criteria followed to create Model 3 can be summarised as:

- Use a single OWL Class to represent:
  - The entire "Fault" Domain Class Hierarchy (which in this case is composed of only one class),
  - The domain for all properties related to the "Fault" concept in the model and
  - The class for all "Fault" individuals.

- Use the OWL Property to represent every term in the closed vocabulary produced in step 1.

- Organize the set of OWL Properties as a flat structure. Do not recreate the hierarchical structure of terms in all the classifications produced in step 1.

This could be seen as the most basic model using the OWL Property as the main design element. OWL Properties that in Model 2 follows a property/subproperty relation are now at the same level in Model 3. Because of this reason, to represent the same information for an individual in Model 3, all the super-properties that in Model 2 can be inferred would have to be specifically applied. In this model, the conceptual overlap in the domain concept does not translate into scenarios of multiple inheritance.

The two variations of Model 2 identified, apply to Model 3 as well.

3.2.1.3.1. Variation 1

These additional design characteristics provide Variation 1 to Model 3:

- Use the OWL Datatype Property to represent the terms.

- The range is the built-in XML Schema datatype xsd:boolean.

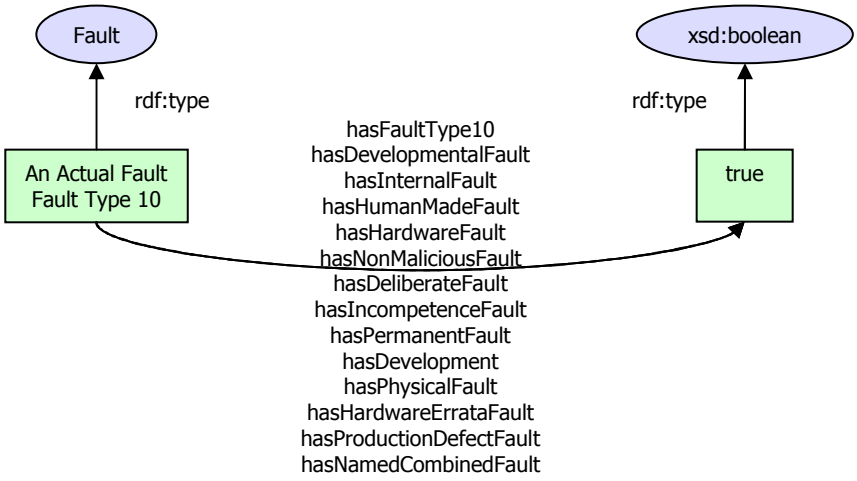

Figure 14. Example of Model 3 Variation 1.

3.2.1.3.2. Variation 2

These additional design characteristics provide Variation 2 to Model 3:

- Use the OWL Object Property to represent the terms.

- The range of all these properties is the OWL Class "Fault".
    a) Using anonymous individuals as property values [Noy, 2005].
    b) Using the same individual as property domain and range value.

The same rationale in Model 2 Variation 2 applies to this variation in Model 3 to favour option b) over option a). Figure 15 only displays the latter.

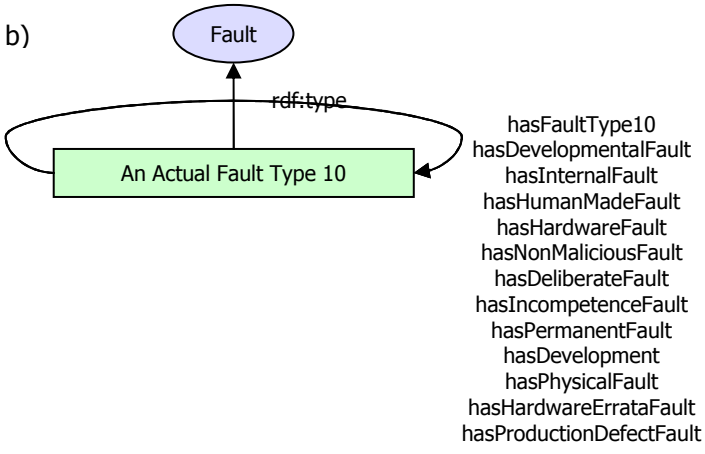

Figure 15. Example of Model 3 Variation 2.

The process of selecting and attaching the applicable properties for an individual in Model 3 could be seen to render certain analogies with Web 2.0 or social Web tagging systems worth exploring. However such analysis at this point is beyond the scope of this report.

3.2.1.4. Summary of internal overlap models

Ultimately, this set of models provides different techniques to represent the same domain concept. Figure 16 shows how the models compare to each other based on the modelling elements they primarily use: class, property and subtype relations.

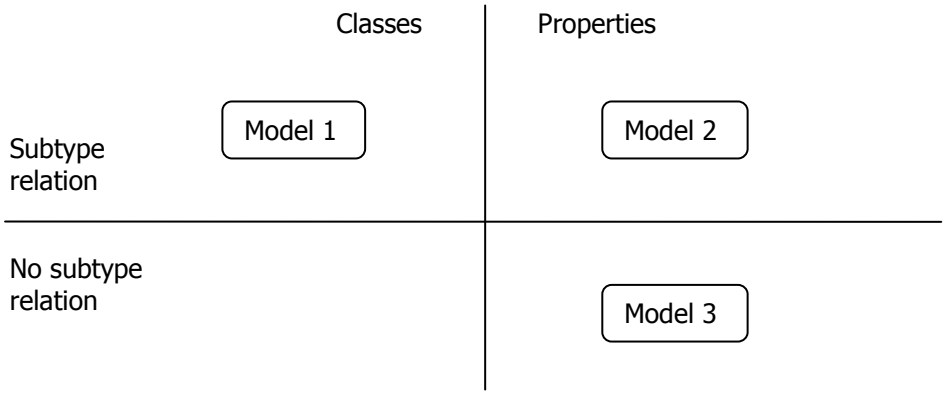

Figure 16. Comparison of proposed ontology models for internal conceptual overlap.

3.2.2. Modelling external overlap

The required entry criteria for all the external conceptual overlap models are the internal overlap models presented in the preceding section and the additional domain concepts that these interact with.

The external models in the following sections provide a low-level view of the high-level ontology design view shown in Figure 10.

The dashed arrow between classes in the figures in this section, indicate that there are additional classes in the "rdfs:subclassOf" transitive relation that have been omitted.

3.2.2.1. Model A

Model A uses Model 1 to represent the Fault domain concept and Approach 4 in [Noy, 2004] to represent the relations between the Fault domain concept and the rest of domain concepts in the overall ReSIST ontology that requires to use Fault as a Value Class Hierarchy.

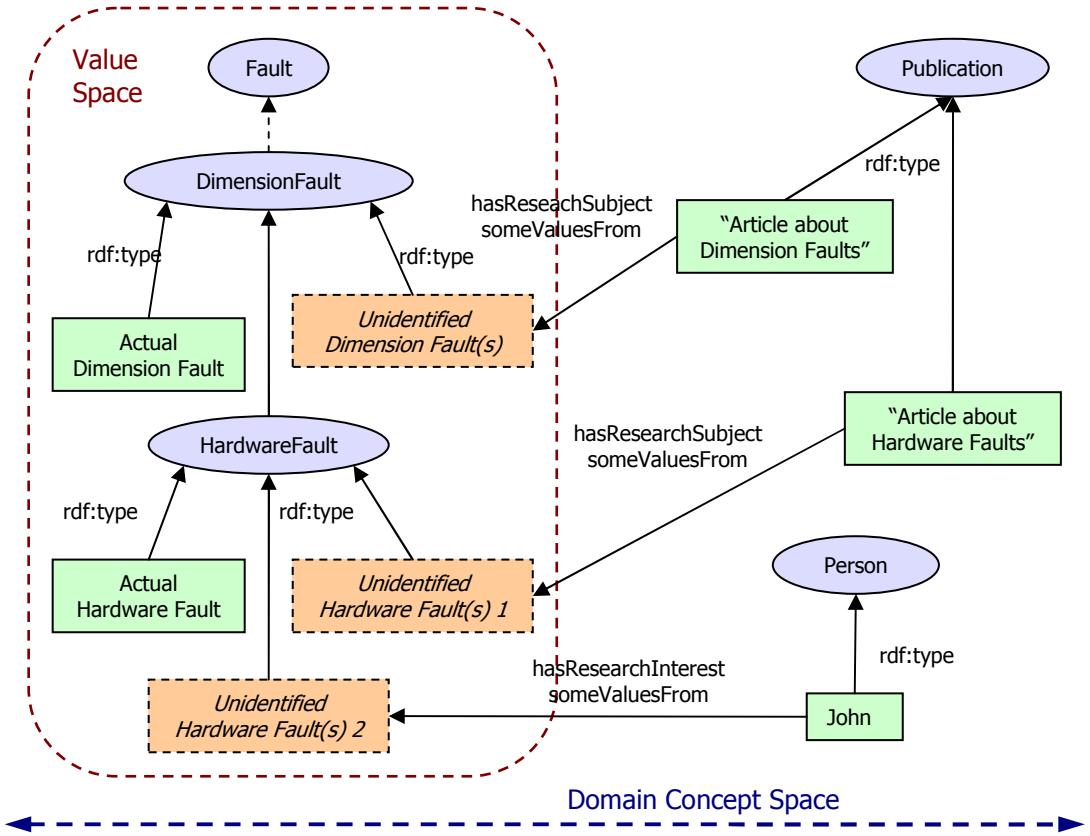

Figure 17. Model A for external conceptual overlap using Model 1.

3.2.2.2. Model B

Model B uses Model 1 to represent the Fault domain concept and the "rdf:type" property to represent the relations between the Fault domain concept and the rest of domain concepts in the overall ReSIST ontology that require to use Fault as a Value Class Hierarchy.

Model B overloads the interpretation of the "rdf:type" property in the ontology because it makes the individuals from the domain concepts external to Fault individuals of Fault as well.

For example: In the case of a Publication individual, the "rdf:type" property is acting as the "hasResearchSubject" property while in the case of a Person individual "rdf:type" is acting as a "hasResearchInterest" property. Figure 18 shows a scenario for this example.

This model can be argued from a design point of view because this representation is not true in the real world. A Publication or a Person is not a type of Fault. It is unintuitive to think of the individual "John" as being a "LogicBombFault" or an "InternalFault" simply because "John" has a research interest in these types of faults.

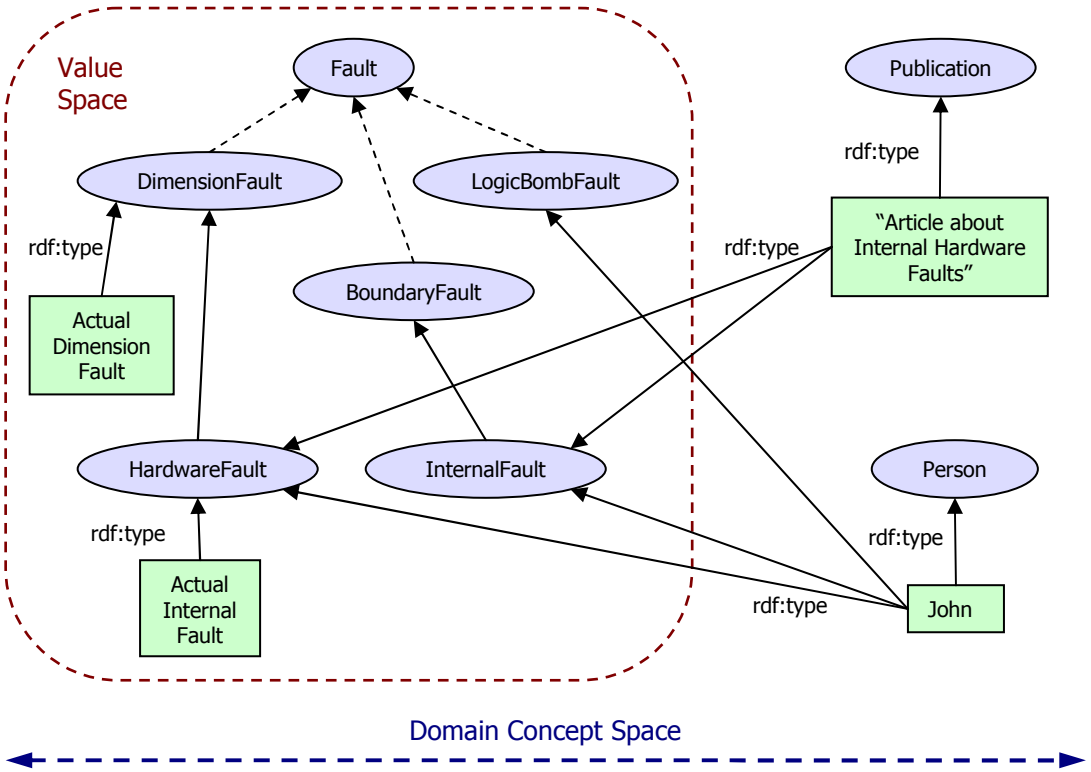

Figure 18. Model B for external conceptual overlap using Model 1.

3.2.2.3. Model C

Model C uses Model 2 to represent the Fault domain concept and the relations between Fault and the rest of domain concepts in the overall ReSIST ontology that require using Fault as a Value Class Hierarchy.

Model C overloads the interpretation of the properties defined by Model 2. For example, the property "hasInternalFault" applied to an individual of "Publication" should be interpreted as the property "hasResearchSubject", while the same property applied to an individual of "Person" should be interpret as "hasResearchInterest".

In that sense, this model can be argued from a design stand point for the same reasons as Model B as well.

However, a possibility to overcome this issue could consist on making this interpretation overload explicit in the model by asserting "hasResearchSubject", "hasResearchInterest" (and similar properties in the same role) as parent properties of the top most properties defined in Model 2. The transitive characteristic of the property/sub-property relation would allow then to interpret "hasInternalFault" (and the rest of properties from Model 2) as type of "hasResearchSubject", "hasResearchInterest", etc, properties. Further analysis of this design approach is required.

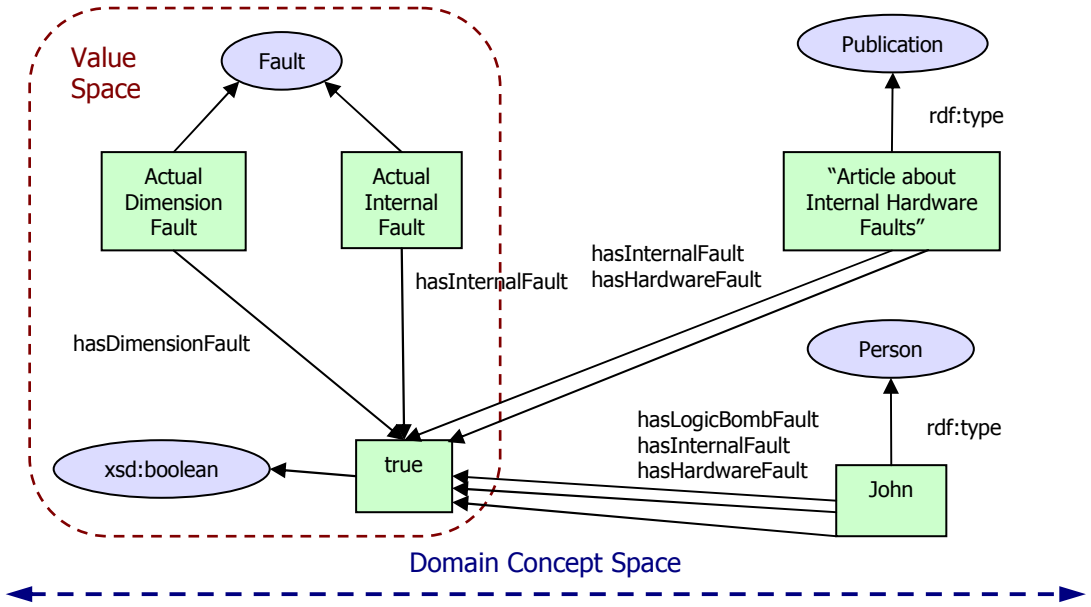

Figure 19. Model C for external conceptual overlap using Model 2 Variation 1.

3.2.2.4. Model D

Model D uses Model 3 to represent the Fault domain concept and the relations between Fault and the rest of domain concepts in the overall ReSIST ontology that require using Fault as a Value Class Hierarchy.

The use of Model 3 in Model D implies that the same observations made between Model 2 and Model 3 can be extrapolated between Model C and Model D.

In addition, the same issue of property interpretation overload considered in Model C applies to Model D as well.

3.2.2.5. Summary of external overlap models

In summary, these are at the moment all the ontology models developed to handle the conceptual overlap among multiple facets considered in the domain concept at both the internal and external scope. There might be though, additional models or additional variations to the existing ones that have not been captured here. In such case, they can be incorporated to this step at a later stage while the rest of this methodology can be reapplied to the new models or variations as usual.

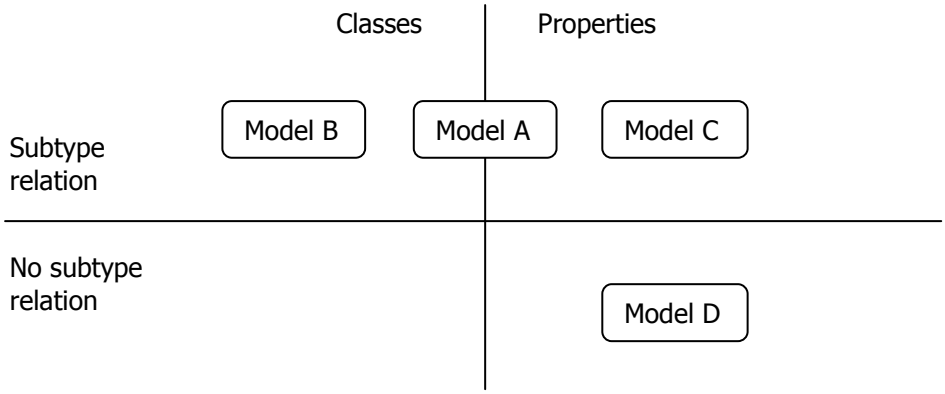

Figure 20. Comparison of proposed ontology models for external conceptual overlap.

The set of all models proposed corresponds to the entry criteria for step 3.

Step 3. Populate same set of individuals in all models

This step creates all the individuals for all proposed models. The individuals are created based on the following design criteria:

- Populate at least one individual for every term in the classification hierarchy obtained in step 1 in order to cover all types of individuals that the model can represent.

- Populate the same set of individuals in all models in order to produce a common ground in which models can be compared to each other.

This approach allows the ontology designer to compare what it entails to populate every type of individual in every model and the differences across models when representing the same individual.

Continuing with the example of the term "Fault Type 10", what follows is the creation of an individual, "FaultType10Ind", across all models using N3 notation [Berners-Lee, 2000].

All examples below assume the following set of prefixes already defined:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix :    <http://www.resist-noe.eu/ontologies/> .
```

Model 1:

```
:FaultType10Ind   rdf:type   :FaultType10 .
```

Mode1 2 variation 1:

```
:FaultType10Ind   rdf:type          :Fault ;
                  :hasFaultType10   “true”^^xsd:boolean .
```

Model 2 variation 2a):

```
:FaultType10Ind   rdf:type   :Fault ,
                             [ a  owl:Restriction;
                               owl:onProperty :hasFaultType10;
                               owl:someValuesFrom : Fault ] .
```

Model 2 variation 2b):

```
:FaultType10Ind   rdf:type          :Fault ;
                  :hasFaultType10   :FaultType10Ind .
```

Mode1 3 variation 1:

```
:FaultType10Ind   rdf:type                 :Fault ;
                  :hasFaultType10          “true”^^xsd:boolean ;
                  :hasDevelopmentalFault   “true”^^xsd:boolean ;
                  :hasInternalFault        “true”^^xsd:boolean ;
                  :hasHumanMadeFault       “true”^^xsd:boolean ;
                  :hasHardwareFault        “true”^^xsd:boolean ;
                  :hasNonMaliciousFault    “true”^^xsd:boolean ;
                  :hasDeliberateFault      “true”^^xsd:boolean ;
                  :hasIncompetenceFault    “true”^^xsd:boolean ;
                  :hasPermanentFault       “true”^^xsd:boolean ;
                  :hasDevelopment          “true”^^xsd:boolean ;
                  :hasPhysicalFault        “true”^^xsd:boolean ;
                  :hasHardwareErrataFault     “true”^^xsd:boolean ;
                  :hasProductionDefectFault   “true”^^xsd:boolean .
```

Model 3 variation 2a):

```
:FaultType10Ind   rdf:type   :Fault ,

                          [ a  owl:Restriction;
                            owl:onProperty :hasFaultType10;
                            owl:someValuesFrom : Fault ] ,

                          [ a  owl:Restriction;
                            owl:onProperty :hasDevelopmentalFault;
                            owl:someValuesFrom : Fault ] ,

                          [ a  owl:Restriction;
                            owl:onProperty :hasInternalFault;
                            owl:someValuesFrom : Fault ] ,
```

```
                          [ a  owl:Restriction;
                            owl:onProperty :hasHumanMadeFault;
                            owl:someValuesFrom : Fault ] ,

                          [ a  owl:Restriction;
                            owl:onProperty :hasHardwareFault;
                            owl:someValuesFrom : Fault ] ,

                          [ a  owl:Restriction;
                            owl:onProperty :hasNonMaliciousFault;
                            owl:someValuesFrom : Fault ] ,

                          [ a  owl:Restriction;
                            owl:onProperty :hasDeliberateFault;
                            owl:someValuesFrom : Fault ] ,

                          [ a  owl:Restriction;
                            owl:onProperty :hasIncompetenceFault;
                            owl:someValuesFrom : Fault ] ,

                          [ a  owl:Restriction;
                            owl:onProperty :hasPermanentFault;
                            owl:someValuesFrom : Fault ] ,

                          [ a  owl:Restriction;
                            owl:onProperty :hasDevelopment;
                            owl:someValuesFrom : Fault ] ,

                          [ a  owl:Restriction;
                            owl:onProperty :hasPhysicalFault;
                            owl:someValuesFrom : Fault ] ,

                          [ a  owl:Restriction;
                            owl:onProperty :hasHardwareErrataFault;
                            owl:someValuesFrom : Fault ] ,

                          [ a  owl:Restriction;
                            owl:onProperty :hasProductionDefectFault;
                            owl:someValuesFrom : Fault ] .
```

Model 3 variation 2b):

```
:FaultType10Ind   rdf:type                    :Fault ;
                  :hasFaultType10             :FaultType10Ind ;
                  :hasDevelopmentalFault      :FaultType10Ind ;
                  :hasInternalFault           :FaultType10Ind ;
                  :hasHumanMadeFault          :FaultType10Ind ;
                  :hasHardwareFault           :FaultType10Ind ;
                  :hasNonMaliciousFault       :FaultType10Ind ;
                  :hasDeliberateFault         :FaultType10Ind ;
                  :hasIncompetenceFault       :FaultType10Ind ;
                  :hasPermanentFault          :FaultType10Ind ;
                  :hasDevelopment             :FaultType10Ind ;
                  :hasPhysicalFault           :FaultType10Ind ;
                  :hasHardwareErrataFault     :FaultType10Ind ;
                  :hasProductionDefectFault   :FaultType10Ind .
```

At the end of this step, all ontology models under consideration have been populated with the same set of individuals, meeting this way the entry criteria to the following step.

Step 4. Define a suite of user questions for all models

This step creates a set of user questions for all proposed models. The questions are created based on the following design criteria:

- The same set of questions should be used in all models.
- The questions should exhaust all different overlapping scenarios represented in the ontology.
- The questions should be able to retrieve different sub-graphs at different hierarchical levels from the classification hierarchies produces in step 1.

This approach allows the ontology designer to compare what it entails to retrieve every type of individual in every model and the differences across models when retrieving the same individual. The rationale for it is derived from two practices:

- Ontology competency questions as described in [Gruninger and Fox, 1995].
- Software unit-testing, in particular the notion of path coverage analysis, common in the traditional Software Engineering development process [CMMI Product Team, 2006]. An initial attempt to adapt unit-testing to the ontology development field can be found in [Vrandecic and Gangemi, 2006].

In the case of ReSIST, there is an additional requirement for the suite of user questions that should be met:

- Users questions that did not retrieve any results should be able to be broadened to retrieve some results as close to the initial request as possible.

Models 2 and 3, based on properties, attempt to address the difficulties in SPARQL to retrieve hierarchy sub-graphs filtering out individuals that otherwise will be part of the set of results because of the transitive characteristic of the "rdfs:subclassOf" relation. These difficulties are a consequence of the limitations in SPARQL to handle the non-monotonic inference rule Negation as Failure (NaF)[3].

[3] See Jena Semantic Web developers mailing list:
http://tech.groups.yahoo.com/group/jena-dev/message/27962
http://tech.groups.yahoo.com/group/jena-dev/message/26866
http://tech.groups.yahoo.com/group/jena-dev/message/26006
http://tech.groups.yahoo.com/group/jena-dev/message/24833
http://tech.groups.yahoo.com/group/jena-dev/message/22528
http://tech.groups.yahoo.com/group/jena-dev/message/22408
http://tech.groups.yahoo.com/group/jena-dev/message/19775
http://tech.groups.yahoo.com/group/jena-dev/message/26997

Once again, continuing with the example of the term “Fault Type 10”, what follows are instances of SPARQL queries that would retrieve the individual “FaultType10Ind” across all models.

All examples below assume the following set of prefixes already defined:

```
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:  <http://www.w3.org/2002/07/owl#>
PREFIX resist: <http://www.resist-noe.eu/ontologies/>
```

Model 1:

```
SELECT ?ind
WHERE {
   ?ind rdf:type resist:FaultType10Ind .
}
```

Mode1 2 variation 1:

```
SELECT ?ind
WHERE {
   ?ind rdf:type resist:Fault .
   ?ind resist:hasFaultType10 true
}
```

Model 2 variation 2a):

```
SELECT ?ind
WHERE {
   ?ind rdf:type resist:Fault .
   ?ind resist:hasFaultType10 ?x .
   ?x rdf:type resist:Fault
}
```

Model 2 variation 2b):

```
SELECT ?ind
WHERE {
   ?ind rdf:type resist:Fault .
   ?ind :hasFaultType10 ?ind
}
```

Mode1 3 variation 1:

```
SELECT ?ind
WHERE {
   ?ind rdf:type resist:Fault .
   ?ind resist:hasFaultType10 true .
   ?ind resist:hasDevelopmentalFault true .
   ?ind resist:hasInternalFault true .
   ?ind resist:hasHumanMadeFault true .
   ?ind resist:hasHardwareFault true .
   ?ind resist:hasNonMaliciousFault true .
   ?ind resist:hasDeliberateFault true .
   ?ind resist:hasIncompetenceFault true .
   ?ind resist:hasPermanentFault true .
   ?ind resist:hasDevelopment true .
   ?ind resist:hasPhysicalFault true .
   ?ind resist:hasHardwareErrataFault true .
   ?ind resist:hasProductionDefectFault true
}
```

Model 3 variation 2a):

```
SELECT ?ind
WHERE {
   ?ind rdf:type resist:Fault .
   ?ind resist:hasFaultType10 ?x .
   ?ind resist:hasDevelopmentalFault ?x .
   ?ind resist:hasInternalFault ?x .
   ?ind resist:hasHumanMadeFault ?x .
   ?ind resist:hasHardwareFault ?x .
   ?ind resist:hasNonMaliciousFault ?x .
   ?ind resist:hasDeliberateFault ?x .
   ?ind resist:hasIncompetenceFault ?x .
   ?ind resist:hasPermanentFault ?x .
   ?ind resist:hasDevelopment ?x .
   ?ind resist:hasPhysicalFault ?x .
   ?ind resist:hasHardwareErrataFault ?x .
   ?ind resist:hasProductionDefectFault ?x .
   ?x rdf:type resist:Fault
}
```

Model 3 variation 2b):

```
SELECT ?ind
WHERE {
   ?ind rdf:type resist:Fault .
   ?ind resist:hasFaultType10 ?ind .
   ?ind resist:hasDevelopmentalFault ?ind .
   ?ind resist:hasInternalFault ?ind .
   ?ind resist:hasHumanMadeFault ?ind .
   ?ind resist:hasHardwareFault ?ind .
   ?ind resist:hasNonMaliciousFault ?ind .
   ?ind resist:hasDeliberateFault ?ind .
   ?ind resist:hasIncompetenceFault ?ind .
   ?ind resist:hasPermanentFault ?ind .
   ?ind resist:hasDevelopment ?ind .
```

```
   ?ind resist:hasPhysicalFault ?ind .
   ?ind resist:hasHardwareErrataFault ?ind .
   ?ind resist:hasProductionDefectFault ?ind
}
```

The definition of this set of user questions across all models according to the design characteristics outlined, meets the exit criteria to move on to the next step.

Step 5. Select an evaluation framework for all models

This step involves the selection and application of an evaluation framework to all proposed models. The application of the same evaluation framework provides a measure of how every model performs compared to each other for the parameters to be considered.

Ideally, the selection process would consist in examining all the different evaluation frameworks available and choosing the one or those that address the parameters intended to be measured in the proposed models. Ontology evaluation is a broad research area in itself and the main approaches in the field are summarized in [Vrandecic, 2006]. This examination remains as a pending task and at the moment postponed for future work.

For the purpose of this report the evaluation framework was derived from the documents released by the World Wide Web Consortium (W3C) Semantic Web Best Practices and Deployment Working Group (SWBPD-WG)[4] by bringing together the superset of all evaluation considerations made throughout the documents ([Noy, 2004] [Rector, 2005] [Noy and Rector, 2006] [Rector and Welty, 2005]).

The evaluation considerations made by the authors varied from pattern to pattern and from document to document. Therefore, for each point, an evaluation category was identified. The evaluation framework presented below, is made of the superset of the main evaluation topics grouped by the categories they belong to:

- Ontology creation:
  - Simplicity. Intuitiveness of the model (using a class as a value may seem unintuitive).
  - Use of restrictions, light-weight or heavy-weight ontology.

- Ontology reuse:
  - Consistency in the interpretation of the ontology.
  - Interoperability with other applications.
  - Interoperability with databases.

- Ontology maintenance and evolution:
  - Maintenance of custom logic needed in the application to fulfil the ontology goals.
  - Maintenance of possible duplication of concept hierarchies.

[4] http://www.w3.org/2001/sw/BestPractices/

    - Maintenance of consistency between hierarchy of classes and (or) individuals.
    - Impact of modification to hierarchies, further partitioning or alternative partitioning of domain concept hierarchies.

- Ontology reasoning:
    - Capability for applications using the model to reason.
    - What can or cannot a generic Description Logics reasoner infer?
    - What would require knowledge of custom additional logic?
    - Inference of transitive relations.

- Ontology expressivity: RDFS, OWL Lite, OWL DL, OWL Full. At the moment, OWL Full ontology models are not in the scope of the ReSIST project although this is might change in future work.

There is an additional evaluation criterion directly linked to step 4 that is not treated in any of the documents from the SWBPD-WG which is decisive in the context of the ReSIST project:

- Ontology querying (knowledge extraction). The goal is to exercise the suite of user questions developed during step 4 in all proposed ontology models. This will provide a framework for comparison of the capabilities to handle user questions across models.

In the case of ReSIST, the evaluation framework has to favour functionality of the application to the end user over correctness of the ontology model. Therefore, the results derived from the ontology querying effort are crucial to determine which model should be selected to model the domain concept of Fault. This results and the rest from the evaluation aspects presented above are the exit criteria of this step and the entry criteria for step 6 in this methodology.

Step 6. Analyze results for all models

This step requires all results derived from the evaluation framework in the previous step of this methodology to be known and available. Based on those results, the expectation is that at least one of the candidate ontology models that participated in the evaluation will exhibit the desired characteristics with respect to a particular evaluation parameter(s).

For ReSIST, the most decisive evaluation parameter is the performance of the models with respect to the set of user questions defined in step 4 of this methodology.

4. Conclusions

This report focused on the practical modelling of ontology domain concepts that can be defined according to multiple facets and the conceptual overlap that occurs among them.

The notion of conceptual overlap and facet has been defined, together with their relation to scenarios of multiple inheritance in the context of modelling ontology domain concepts.

A review of research areas relevant to the problem of addressing conceptual overlap has been presented. These include ontology modelling, ontology design patterns, analysis of multiple inheritance in object-oriented programming languages and ontology evaluation. Opportunities for improvement in the current methodologies to address this specific problem have been identified.

Similarities have been elicited between two ontology modelling design patterns that share how they use anonymous individuals to provide the value for ontology properties. These similarities allow expanding the notion of "value partition" to other structures of domain concept hierarchies. A terminology is introduced to capture the ontology modelling elements involved in this extension of "value partition". A series of conclusions from certain characteristics among these elements are drawn. These findings lead to the characterization of two types of conceptual overlap: internal and external to the domain concept under studied. The first one occurs inside the domain concept and the second occurs across the domain concept and additional concepts in the ontology.

These considerations make *explicit* some of the *implicit* modelling decisions taken previously in the ontology development field. Our contribution is proven with the representation of the conceptual overlap in the "Fault" domain concept that is part of the ReSIST[5] project.

It is also put forward, an ontology modelling methodology to address this problem in a structured manner. The methodology comprises a series of steps. For every step, an entry and an exit criteria has been defined that should be met before starting or finishing respectively such step.

As part of one of the steps in the methodology, several alternative ontology models have been presented to handle internal and external conceptual overlap and different guidelines have been provided to populate, query, and evaluate the candidate ontology models. In this sense, this methodology provides a generic framework to compare ontology models intended to represent the same domain concept that for the purpose of this report is used to compare different alternatives to address conceptual overlap.

[5] http://www.resist-noe.org/

5. Future Work

Currently, there are prototypes developed of the internal conceptual overlap Models 1, 2 and 3 presented in section 4, available online via a source code repository and project management tool[6].

The prototypes have been developed using the Jena[7] Semantic Web development framework for Java, and they include full implementation of up to step 3 and partial implementation of step 4 in the methodology discussed in section 4.

On that basis, the different lines open for future research can be grouped mainly in two. The first one deals with completing the remaining steps of the methodology while the second would attempt to introduce enhancements to it.

Completion of the remaining steps, step 4 and beyond, could originate certain subtasks which might lead to research paths not explored so far such as:

- Use the notion of path coverage analysis that is part of unit testing practices in traditional Software Engineering to define the suite of user questions in step 4 [Vrandecic and Gangemi, 2006]. The idea being, attempting to cover as many conceptual overlap scenarios across candidate models as possible.

- Study the limitations of SPARQL to handle queries involving Negation as Failure (NaF) to retrieve sub-graphs from the target ontology models.

- Survey of the current state of the art in ontology evaluation to identify the evaluation method or methods, if any, that could be employed to measure the candidate models against the parameter or parameters that want to be considered in relation to the issue of conceptual overlap. The result of such survey may well conclude that the current ontology evaluation tools are not fit for the required purpose and the creation of new ones may have to be considered.

The second line of further research focuses on enhancements to the methodology presented in section 4 that can be identified at this point. These enhancements include:

- Characterize the design criteria for a model that combines OWL Classes and Properties to represent internal conceptual overlap.

- Characterize the design criteria of external conceptual overlap models.

- Identify what parts of the methodology are specific to the problem of conceptual overlap and what are generic to compare ontology models intended to represent the same domain concept.

- Formalize the characterization of the exit criteria and entry criteria of the different steps in the method.

[6] http://br205r-owlmi.ugforge.ecs.soton.ac.uk/
[7] http://jena.sourceforge.net/

- Opportunities for automation of certain subtasks in some of the steps in the proposed methodology. Provided that the different modelling design patterns and the different entry and exit criteria are characterized in detail, an application framework such as Jena could develop for example the different candidate ontology models, the creation of individuals and the set of user questions.

An additional task worth considering that would help to consolidate the principles established throughout this report is the application of such principles to other examples of domain concepts that meet the characteristics of conceptual overlap laid out here.

In an attempt to foresee completion of the different activities within the time window available for the completion of this PhD program, a tentative work plan for the remaining year could look as follows:

By the end of April 2008, roughly 4 months after the end of this report, the rest of steps for the "Fault" domain concept in ReSIST should be completed, and a full iteration of the methodology presented should be executed on the different candidate ontology models. The results obtained should be recorded and presented. This time would provide about two months to find solutions to the obstacles outlined earlier in defining a suite of user questions, and another two months to identify an ontology evaluation tool as per the characteristics demanded in step 5 of the methodology.

By the end of July 2008, 3 months later, at least two additional examples of conceptual overlap in concepts from different domains should be proposed and once again, an iteration over the methodology to best model such domain concepts should be carried out. The results from these additional examples should be contrasted among each other and with those from the "Fault" domain concept.

The findings obtained along the process described should serve as the basis to document the final thesis due at the end of December 2008.

Acknowledgments

This work is supported under the ReSIST Network of Excellence, which is sponsored by the Information Society Technology (IST) priority in the EU Sixth Framework Programme (FP6) under contract number IST 4 026764 NOE.

Additionally, we are thankful to the following people for their comments and feedback throughout this work: Hugh Glaser, Afraz Jaffri, Ian Millard, Madalina Croitoru, Harith Alani, Yannis Kalfoglou, Asuncion Gomez-Perez, Andy Seaborne, Ian Horrocks, Brandon Ibach, Bijan Parsia and Alan Rector.

References

[ACM, 1998]
ACM (1998) The ACM Computing Classification System. Version valid in 2002, http://www.acm.org/class/1998/

[AKT, 2002]
AKT (2002) The AKT reference ontology. http://www.aktors.org/publications/ontology/

[Alani et al., 2006]
Alani H, Harris S, O'Neil B (2006) Winnowing Ontologies based on Application Use. In: Proceedings of 3rd European Semantic Web Conference (ESWC), Budva, Montenegro

[Anderson et al., 2007]
Anderson, T., Andrews, Z., Fitzgerald, J., Randell, B., Glaser, H. and Millard, I. (2007) The ReSIST Resilience Knowledge Base. In Proceedings of DSN 2007 - The 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Edinburgh, UK

[Avizienis et al., 2005]
Avizienis A, Laprie JC, Randell B, Landwehr C (2005) Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing, 1(1):11--33

[Beckett, 2007]
Beckett D (2007) Turtle - Terse RDF Triple Language. 20 November 2007. http://www.dajobe.org/2004/01/turtle/

[Berners-Lee et al., 2001]
Berners-Lee T, Hendler J, Lassila O (2001) The semantic web. Scientific American

[Berners-Lee, 2000]
Berners-Lee T (2000) Primer: Getting into RDF and Semantic Web using N3. http://www.w3.org/2000/10/swap/Primer

[Berners-Lee, 1998]
Berners-Lee T (1998) Semantic Web Roadmap. World Wide Web Consortium (W3C) http://www.w3.org/DesignIssues/Semantic.html/

[Blomqvist, 2007]
Blomqvist E (2007) OntoCase - A Pattern-based Ontology Construction Approach. To appear in: Proccedings of OTM 2007: ODBASE - The 6th International Conference on Ontologies, DataBases, and Applications of Semantics, Vilamoura, Algarve, Portugal, November 25-30, 2007

[Blomqvist and Sandkuhl, 2005]
Blomqvist E and Sandkuhl K (2005) Patterns in Ontology Engineering – Classification of Ontology Patterns. In: Proc. of 7th International Conference on Enterprise Information Systems, Miami, USA, May 2005

[Brase and Nejdl, 2003]
Brase J and Nejdl W (2003) Ontologies and Metadata for eLearning. Springer Verlag, pp 579-598

[Cargill 1991]
Cargill T (1991) The case against multiple inheritance in C++. USENIX Computing Systems, 4(1):69-82, Winter 1991

[CETIS, 2004]
CETIS (2004) UK Learning Object Metadata Core Draft 0.2. Centre for Educational Technology Interoperability Standards. University of Bolton. Bolton, UK. http://www.cetis.ac.uk/profiles/uklomcore/uklomcore_v0p2_may04.doc/

[CMMI Product Team, 2006]
CMMI Product Team (2006) "CMMI for Development, Version 1.2", CMU/SEI-2006-TR-008, Software Engineering Institute, Carnegie Mellon University

[Connolly and Begg, 1998]
Connolly T, Begg C (1998) Database Systems: A Practical Approach to Design, Implementation, and Management. 2nd Ed. Addison-Wesley, Harlow, England

[d'Aquin et al., 2007a]
d'Aquin M, Schlicht A, Stuckenschmidt H, Sabou M (2007) Ontology Modularization for Knowledge Selection: Experiments and Evaluations. 18th International Conference on Database and Expert Systems Applications - DEXA '07, Regensburg, Germany

[d'Aquin et al., 2007b]
d'Aquin M, Baldassarre C, Gridinoc L, Angeletou S, Sabou M, Motta E (2007) Characterizing Knowledge on the Semantic Web with Watson. Workshop: Evaluation of Ontologies and Ontology-based tools, 5th

International EON Workshop, International Semantic Web Conference (ISWC'07), Busan, Korea

[Dean and Schreiber, 2004]
Dean M, Schreiber G, (eds) (2004) OWL Web Ontology Language Reference. W3C Recommendation

[Ehrig et al., 2004]
Ehrig M, Gabel T, Haase P, Sure Y, Tempich C, Voelker J (2004) Use Cases. SEKT: Semantically Enabled Knowledge Technologies. IST-2003-506826 Project Deliverable 7.1.1.a

[Fernandez-Lopez et al., 2002]
Fernandez-Lopez M (ed) (2002) A survey on methodologies for developing, maintaining, evaluating and reengineering ontologies. OntoWeb IST-2000-29243 Project Deliverable 1.4

[Fernandez-Lopez et al., 1997]
Fernandez-Lopez M, Gomez-Perez A, Juristo N (1997) METHONTOLOGY: From Ontological Art Towards Ontological Engineering. Spring Symposium on Ontological Engineering of AAAI. Stanford University, California, pp 33-40

[Gamma et al., 1995]
Gamma E, Helm R, Johnson R and Vlissides J (1995). Design Patterns: Elements of Reusable Object-Oriented Software, hardcover, 395 pages, Addison-Wesley. ISBN 0-201-63361-2

[Gangemi, 2005]
Gangemi A (2005) Ontology Design Patterns for Semantic Web Content. Proceedings ISWC 2005, LNCS 3729, pp 262-276

[Glaser et al., 2007]
Glaser, H., Millard, I., Rodriguez-Castro, B. and Jaffri, A. (2007) Demonstration: Knowledge-Enabled Research Infrastructure (Poster). In Proceedings of 4th European Semantic Web Conference, Innsbruck, Austria

[Glaser et al., 2004]
Glaser H, Alani H, Carr L, Chapman S, Ciravegna F, Dingli A, Gibbins N, Harris S, schraefel, mc, Shadbolt N (2004) CS AKTiveSpace: Building a semantic web application. In: Bussler C, Davies J, Fensel D, Studer R, (eds) ESWS. Volume 3053 of Lecture Notes in Computer Science. Springer, pp 417–432

[Gomez-Perez et al., 2004]
Gomez-Perez A, Fernandez-Lopez M, Corcho O (2004) Ontological Engineering. Springer Verlag, London

[Good et al., 2006]

Good, B.M., Tranfield, E.M., Tan, P.C., Shehata, M., Singhera, G.K., Gosselink, J. and Wilkinson, M.D. (2006) Fast, cheap and out of control: A zero curation model for ontology development. In Pacific Symposium on Biocomputing, (Hawaii, USA, 2006), 128--139

[Gruninger and Fox, 1995]
Gruninger M, Fox MS (1995) Methodology for the design and evaluation of ontologies. In: Skuce D (ed) IJCAI95 Workshop on Basic Ontological Issues in Knowledge Sharing. Montreal, Canada, pp 6.1-6.10

[Guarino and Welty, 2002]
Guarino N and Welty C (2002) Evaluating Ontological Decisions with OntoClean. In: Communications of the ACM, 45 (2) pp 61-65

[Holi and Hyvönen, 2005]
Holi M and Hyvönen E (2005) Modeling Degrees of Overlap in Semantic Web Ontologies. Proceedings of the ISWC Workshop Uncertainty Reasoning for the Semantic Web (Paulo C. G. da Costa, Kathryn B. Laskey, Kenneth J. Laskey and Michael Pool (eds.)), CEUR Workshop Proceedings, Galway, Ireland, Nov, 2005

[Horridge et al., 2004]
Horridge M, Knublauch H, Rector A, Stevens R, Wroe C (2004) Practical Guide To Building OWL Ontologies Using the Protégé-OWL Plugin and CO-ODE Tools. Technical Report, Ed. 1.0, The University Of Manchester

[IEEE, 2002]
IEEE (2002). Draft Standard for Learning Object Metadata. Sponsored by the IEEE Learning Technology Standards Committee. IEEE 1484.12.1-2002. http://ltsc.ieee.org/wg12/files/LOM_1484_12_1_v1_Final_Draft.pdf

[Kingston, 2001]
Kingston J (2001) Ontologies, Multi-Perspective Modelling and Knowledge Auditing. In: Ontologies Workshop at the Second German/Austrian Conference on Artificial Intelligence (KI-2001)

[Knublauch et al., 2006]
Knublauch H, Oberle D, Tetlow P, Wallace E, (2006) A Semantic Web Primer for Object-Oriented Software Developers. W3C Working Group Note 9 March 2006. http://www.w3.org/TR/sw-oosd-primer/

[Manola and Miller, 2004]
Manola F, Miller E (2004) RDF Primer. W3C Recommendation. http://www.w3.org/TR/rdf-primer/

[McGuinness, 2001]
McGuinness DL (2001) Ontologies come of age. In: Fensel D et al (eds) Spinning the Semantic Web: Bringing the World Wide Web to its Full Potential. MIT Press, Cambridge, MA

[Millard et al., 2006]
Millard I, Jaffri A, Glaser H, Rodriguez B (2006) Using a Semantic MediaWiki to Interact with a Knowledge Based Infrustructure (Poster). Submitted to 15th International Conference on Knowledge Engineering and Knowledge Management. Podebrady, Czech Republic

[Motta and Sabou 2006]
Motta E, Sabou M (2006) Next Generation Semantic Web Applications. 1st Asian Semantic Web Conference. Beijing, China

[Nilsson et al., 2003]
Nilsson M, Palmer M, Brase J (2003) The LOM RDF binding – principles and implementation. The 3rd Annual Ariadne Conference, 20-21 November 2003, Belgium

[Noy, 2004]
Noy N, (2004) Representing Classes As Property Values on the Semantic Web. W3C Working Group Note 5 April 2005. http://www.w3.org/TR/swbp-classes-as-values/

[Noy and Klein, 2002]
Noy NF, Klein M (2002) Ontology Evolution: Not the Same as Schema Evolution. In: Stanford Medical Informatics Technical Report SMI-2002-0926. Stanford, California

[Noy and McGuinness, 2001]
Noy N, McGuinness DL, (2001) Ontology development 101: A guide to creating your first ontology. In: Technical Report KSL-01-05, Stanford Knowledge Systems Laboratory. Stanford, California

[Noy and Musen, 2000]
Noy NF, Musen MA (2000) PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In: Rosenbloom P, Kautz HA, Porter B, Dechter R, Sutton R, Mittal V (eds) 17th National Conference on Artificial Intelligence (AAAI'00). Austin, Texas, pp 450-455

[Noy and Rector, 2006]
Noy N, Rector A, (2006) Defining N-ary Relations on the Semantic Web. W3C Working Group Note 12 April 2006. http://www.w3.org/TR/swbp-n-aryRelations/

[Pan et al., 2007]
Pan J.Z., Lancieri L., Maynard D., Gandon F., Cuel R and Leger A. (2007) Knowledge Web Deliverable D1.4.2.v2. Success Stories and Best Practices. January 2007. Available at: http://www.csd.abdn.ac.uk/~jpan/pub/TR/D142v2-final.pdf

[Powers, 2003]
Powers S (2003) Practical RDF. O'Reilly & Associates. ISBN 0-596-00263-7

[Prieto-Diaz, 2003]
Prieto-Diaz R (2003) A Faceted Approach to Building Ontologies. In: IEEE International Conference on Information Reuse and Integration. IEEE Computer Society Press, pp. 458-465

[Prudhommeaux and Seaborne, 2005]
Prudhommeaux E, Seaborne A (2005) A SPARQL Query Language for RDF. W3C Working Draft. http://www.w3.org/TR/rdf-sparql-query/

[Rector, 2005]
Rector A, (2005) Representing Specified Values in OWL: "value partitions" and "value sets". W3C Working Group Note 17 May 2005. http://www.w3.org/TR/swbp-specified-values/

[Rector and Welty, 2005]
Rector A and Welty C (2005) Simple part-whole relations in OWL Ontologies. W3C Editor's Draft 11 Aug 2005. http://www.w3.org/2001/sw/BestPractices/OEP/SimplePartWhole/index.html

[Rector et al., 2004]
Rector A.L., Drummond N, Horridge M, Rogers J, Knublauch H, Stevens R, Wang H and Wroe C (2004) OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns. Enrico Motta, Nigel Shadbolt, Arthur Stutt, Nicholas Gibbins (Eds.): Engineering Knowledge in the Age of the Semantic Web, 14th International Conference, EKAW 2004, Whittlebury Hall, UK, October 5-8, 2004, Proceedings. Lecture Notes in Computer Science 3257 Springer 2004, ISBN 3-540-23340-7

[Rector, 2003]
Rector AL (2003) Modularisation of domain ontologies implemented in description logics and related formalisms including OWL. ACM Press, 2003, 121-128

[Rector et al., 2001]
Rector AL, Wroe C, Rogers J and Roberts A (2001) Untangling taxonomies and relationships: personal and practical problems in loosely coupled development of large ontologies. Proceedings of the First International Conference on Knowledge Capture (K-CAP 2001), October 21-23, 2001, Victoria, BC, Canada. ACM 2001, ISBN 1-58113-380-4

[ReSIST, 2006]
The ReSIST Project (2006) Resilience and Survivability in Information Society Technology (IST). IST 4 026764 NOE. http://www.resist-noe.org/

[Rumbaugh et al., 1991]
Rumbaugh J, Blaha M, Premerlani W, Eddy F, Lorensen W (1991) Object-oriented modeling and design. Englewood Cliffs, New Jersey. Prentice Hall

[Shadbolt et al., 2004]

Shadbolt NR, Gibbins N, Glaser H, Harris S, schraefel mc (2004) CS AKTive Space or how we stopped worrying and learned to love the Semantic Web. IEEE Intelligent Systems

[Shirky, 2005]
Shirky C (2005) Ontology is Overrated: Categories, Links and Tags. In: [online] Clay Shirky's Writings About the Internet. http://shirky.com/writings/ontology_overrated.html

[Skuce and Lethbridge, 1995]
Skuce D, Lethbridge TC (1995) CODE4: A Unified System for Managing Conceptual Knowledge. International Journal of Human-Computer Studies 42(4):413-451

[Smith, 2006]
Smith B (2006) Against Idiosyncrasy in Ontology Development. Forthcoming in B. Bennett and C. Fellbaum (Eds.), Formal Ontology and Information Systems, (FOIS 2006), Baltimore November 9—11, 2006

[Spaccapientra et al., 2004]
Spaccapientra S, Parent C, Vangenot C, Cullot N (2004) On Using Conceptual Modeling for Ontologies. In: Proceedings of the Web Information Systems Workshops (WISE 2004 Workshops), Lecture Notes in Computer Science 3307, 22-33

[Spyns et al., 2002]
Spyns P, Meersman R, Jarrar M (2002) Data modelling versus ontology engineering. ACM SIGMOD Rec 31(4):12–17

[Studer et al., 1998]
Studer R, Benjamins VR, Fensel D (1998) Knowledge Engineering: Principles and Methods. IEEE Transactions on Data and Knowledge Engineering 25(1-2):161-197

[Suarez-Figueroa et al., 2007]
Suarez-Figueroa MC, Brockmans S, Gangemi A, Gomez-Perez A, Lehmann J, Lewen H, Presutti V and Sabou M (2007) NeOn Modelling Components. UPM, 2007

[Sure and Studer, 2002]
Sure Y and Studer R (2002) On-To-Knowledge Methodology - Final Version. Institute AIFB, University of Karlsruhe, On-To-Knowledge Deliverable 18, 2002. Available at http://www.aifb.uni-karlsruhe.de/WBS/ysu/publications/OTK-D18_v1-0.pdf

[Tempero and Biddle, 2000]
Tempero E and Biddle R (2000) Simulating Multiple Inheritance in Java. Journal of Information and Software Technology, (55):87-100, 2000

[Uschold and Gruninger, 1996]

Uschold M, Gruninger M (1996) Ontologies: Principles, Methods, and Applications. Knowledge Eng. Rev., Vol. 11, No. 2, pp. 93-155

[Uschold and King, 1995]
Uschold M, King M, (1995) Towards a Methodology for Building Ontologies. In: Skuce D (eds) IJCAI'95 Workshop on Basic Ontological Issues in Knowledge Sharing. Montreal, Canada, pp 6.1-6.10

[Vrandecic, 2006]
Vrandecic D (2006) Ontology Evaluation for the Web - PhD proposal. In Joerg Diederich and Enrico Motta and Elena Paslaru Bontas, Proceedings of the KnowledgeWeb PhD Symposium KWEPSY 2006. Budva, Montenegro, June 2006

[Vrandecic and Gangemi, 2006]
Vrandecic D and Gangemi A (2006) Unit tests for ontologies. Jarrar, M.; Ostyn, C.; Ceusters, W. & Persidis, A. (ed.) Proceedings of the First International Workshop on Ontology content and evaluation in Enterprise, Springer, 2006

[Waldo 1991]
Waldo J (1991) Controversy: The case for multiple inheritance in C++. USENIX Computing Systems, 4(2):157-172, Spring 1991

[Wang et al., 2006]
Wang TD, Parsia B and Hendler J (2006) A Survey of the Web Ontology Landscape. In Proc. of Int. Semantic Web Conference (ISWC 2006), 2006