View Inheritance as an Extension of the Normalization Ontology Design Pattern

Bene Rodriguez-Castro, Hugh Glaser, and Ian Millard

Intelligence, Agents and Multimedia Group,
School of Electronics and Computer Science,
University of Southampton,
Southampton SO17 1BJ, UK,
{b.rodriguez, hg, icm}@ecs.soton.ac.uk

Abstract. There are ontology domain concepts that are difficult to represent due to the complexities in their definition and the presence of multiple alternative criteria to classify their abstractions. To assist ontologists in overcoming these challenges, an analysis of available design patterns in ontology and object-oriented modeling has been carried out. As a result, the View Inheritance Ontology Design Pattern (ODP) is introduced. The pattern extends the Normalization ODP (a.k.a. Untangling or Modularization) and revelas the notion of Inter- and Intra-criterion Multiple Inheritance. Our contribution is illustrated with a concrete example of a use case scenario that benefits from the outcome of this study.

1 Introduction

Ontologies have emerged as one of the key components needed for the realization of the Semantic Web vision. Ontologies bring with them a broad range of development activities that can be grouped into what is called ontology engineering. 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 [1]. Within ontology engineering, this research primarily focuses on ontology modeling, more specifically on Ontology Design Patterns (ODPs) [2] and on how they can help representing complex domain concepts. ODPs have evolved from the general notion of design pattern, defined in [2] as "archetypal solutions to design problems in a certain context" and they are justifiably receiving a significant amount of attention by ontologists due to the preceding success achieved by software design patterns [3].

In this study, we introduce the View Inheritance ODP. The View Inheritance pattern intends to represent ontology domain concepts that presents multiple alternative criteria to classify their abstractions. The motivation that led us to the development of the View Inheritance pattern originates in the ReSIST (Resilience for Survivability in Information Society Technologies) project [4]. One of the objectives of ReSIST is to create a knowledge base application in the field of resilient and dependable computing. The ReSIST Knowledge Base

(RKB)[1] features an ontology in the domain of resilient systems. This ontology was built using the definitions and taxonomies presented in [5].

Among all the ReSIST concepts, one that particularly stands out from a representational point of view is the concept of “Fault”. The concept of “Fault” referred hereto is extensively detailed in [5]. This concept involves certain complexities that makes it difficult to represent in the ReSIST ontology, such as a) the dual role that it supports in the ontology, b) the number of relationships that it participates in with other ontology domain concepts, c) support for classifying occurrences of actual faults in real world systems and d) providing a keyword index for subjects of publications and research interests/areas of projects, institutions or people. The characteristics of role and reusability of domain concepts that we identified in [6] laid out some guidelines to handle the first two aspects. To handle the rest, it is crucial that all types of faults identified in [5] can be represented in the ReSIST ontology.

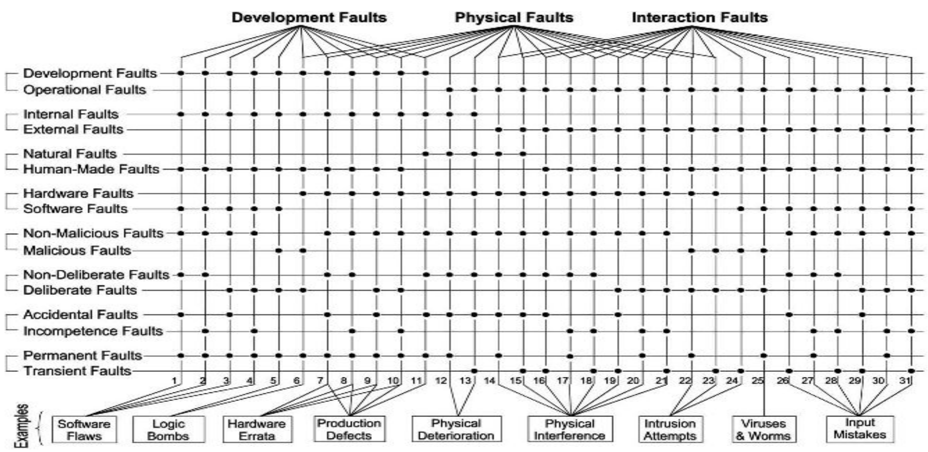

Fig. 1. Matrix representation of “Fault” in Avizienis et al. [5] used in the ReSIST KB ontology

Figure 1 in particular, shows a matrix representation of all types of faults which may affect a system during its life (see also Figure 5b in [5] for a tree representation of these faults). Implicitly, the figure reveals several alternative criteria for the classification of faults:

- A first criterion can be derived from the left column of the matrix. This column represents the values of the eight basic viewpoints (see Figure 4 in [5]) which lead to the elementary fault classes : “Development/Operational Faults”, “Internal/External Faults” and so on.

[1] http://rkbexplorer.com/

– A second criterion can be abstracted from the bottom row (listing numbers 1 to 31). This row represents the 31 likely combinations of fault classes out of the 256 possible.
– A third criterion is implicit at the top row, representing the three major partially overlapping groupings of faults: "Development", "Physical" and "Interaction".
– A fourth criterion can be seen at the bottom row, labeled "Examples", containing nine illustrative examples of fault classes.

The representation of multiple alternative criteria (views) to classify the abstractions of a certain domain concept motivated the development of the View Inheritance ODP, which is explained throughout the paper as follows: Section 2 presents a brief overview of the main work related to the modeling of ODPs. Section 3 introduces the View Inheritance ODP in detail and finally, Section 4 covers the conclusions gathered from this endeavor and open lines for further investigation.

2 Related Research

An initial set of ODPs is introduced in [2]. The notion of Conceptual (or Content) Ontology Design Patterns (CODePs) is defined and several examples are included. Future work in the direction of CODePs has been carried out in the context of the NeOn project[2] [7][8]. They present a more detailed overview and refinement of different types of ODPs at different levels of abstraction (see Figure 2.2 in [8]) and a thorough catalogue of patterns is also documented and made available online[3].

Experiences in the development of large ontologies in the Biology domain led to the development of a separate repository of ODPs[4] [9] although with a significant degree of overlap with the one previously mentioned.

From all the patterns surveyed, only the Normalization ODP [9][10] analyzes the implications of having a high number of multiple inheritance relations in the ontology and it refers to the notion of modelling different *semantic axes* as the cause that can lead to polihierarchical structures or a *tangled* ontology. It then outlines a very effective step by step procedure that would *untangle* the ontology becoming a collection of independent modules easy to maintain.

To achieve this however, it relies on modelling constructs available only at the OWL-DL expressivity level of the OWL language (such as *owl:disjointWith* and to some extent *owl:intersectionOf*). This also implies the need of an OWL-DL capable reasoner in order to fully benefit from the use of this pattern. Unfortunately, in the context of ReSIST support for OWL-DL reasoning is beyond the scope of the project.

[2] http://www.neon-project.org
[3] http://ontologydesignpatterns.org/
[4] http://odps.sourceforge.net/

Another aspect where the Normalization ODP and the representation of "Fault" in ReSIST diverge is connected to the normalization criteria of the pattern. These criteria requires the primitive skeleton of the domain ontology to consist only of disjoint homogeneous trees [10]. Intuitively, this appears as a very limiting constrain considering the amount of overlap that Figure 1 reveals among the types of faults identified.

Praising the virtues of the Normalization pattern, we attempted to find complementary material to characterize the modelling scenario that the many faces of "Fault" presents. In that sense and again in the domain of O-O software, the rationale of View Inheritance given in [11] as part of his detailed taxonomy of *types of inheritance* and discussion of *multiple inheritance* seem to correlate very well with the representation needs of "Fault".

In summary, the related work referred hereto, provided an essential framework that served as the basis for the proposed View Inheritance ODP to assist with the development of the ontology component of ReSIST.

3 View Inheritance Ontology Design Pattern

It might be too early to consider View Inheritance a full right member of the ODP family given that certain aspects of it are still being refined and a more extensive collection of archetypal use cases is still needed. The intent of the pattern is to provide a characterization to the ontological modelling problem of representing a certain domain concept whose abstractions can be classified according to multiple alternative criteria. It represents all of these relevant classification criteria so that multiple possible combinations of the concepts that refine them are allowed.

The applicability requirements below stems from the use of View Inheritance in the O-O design [11]. Nonetheless, the level of abstraction is high enough to deem them suitable for ontology development as well. In ontology design terms, these applicability requirements can be summarized as:

- The various classification criteria of the ontology domain concept being represented are equally important, so any choice of a primary one would be arbitrary.
- Many possible combinations of the concepts that refine the various classification criteria are needed.
- Reusability. The domain concepts under consideration are so important as to justify spending significant time to get the best possible inheritance structure.

Structure (Diagram). Figure 2 provides a generic graphical representation of the View Inheritance pattern while Figure 3 shows a subset of the classes involved in the overall representation of "Fault" and how it aligns to this generic structure.

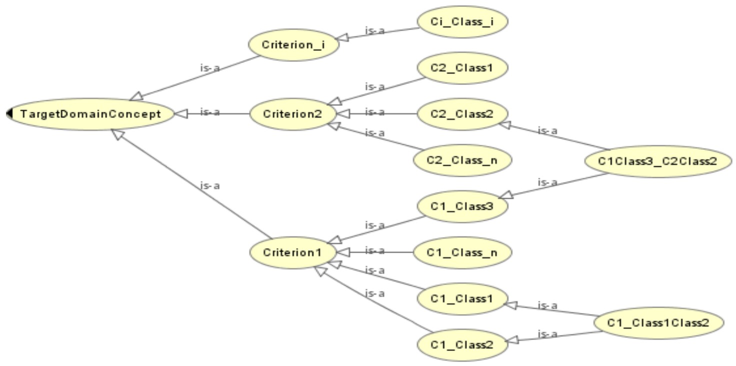

Fig. 2. Structure of a generic use case of the View Inheritance ODP.

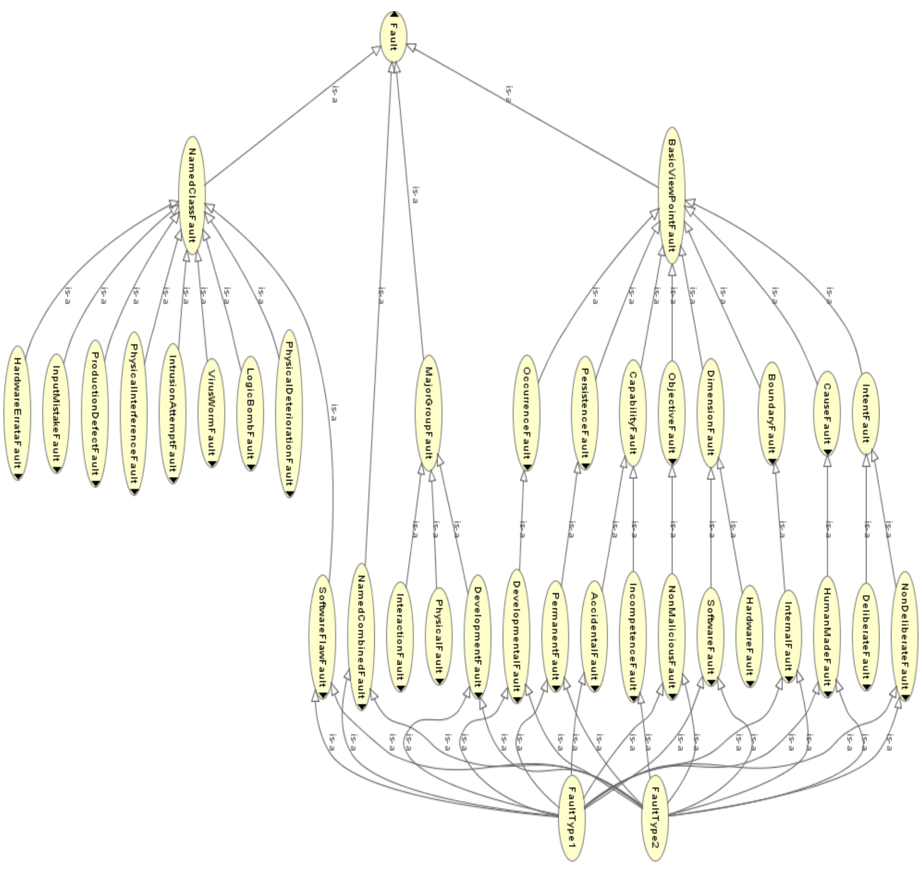

Fig. 3. Structure of the View Inheritance ODP for the representation of "Fault". For simplicity, only 2 types of faults are shown out of the 31 types defined.

Elements and Relationships. The classes participating in the pattern together with their responsibility are listed below. The generic names for classes corresponds to classes located in Figure 2 and the names of classes from the representation of the "Fault" example are located in Figure 3.

- **TargetDomainConcept** (Fault)
  - This class represents the ontology domain concept being defined for which multiple alternative abstraction criteria exist.
- **Criterion_i** (BasicViewPointFault, MajorGroupFault, NamedClassFault, NamedCombinedFault)
  - These classes represent each one of the alternative abstraction criteria of the TargetDomainConcept (Criterion1, Criterion2, Criterion_i in Figure 2). The list of classes may not be exhaustive or pairwise disjoint.
- **Ci_Class_x**[5] (All subclasses of BasicViewPointFault, MajorGroupFault, NamedClassFault, NamedCombinedFault)
  - These classes refine each abstraction criteria class (C1_Class1, ..., C2_Class1, ..., Ci_Class_i in Figure 2). The list of classes may not be exhaustive or pairwise disjoint.
- **CiClass_x_CjClass_y**[6]**, Ci_Class_xClass_y**[7] (FaultType1, FaultType2, ..., FaultType32)
  - These classes participate in multiple inheritance relationships combining different refinements from the alternative abstraction criteria classes (C1Class3_C2Class2 and C1_Class1Class2 in Figure 2).

Inter- and Intra-criterion Multiple Inheritance. There is an interesting feature regarding the types of multiple inheritance relations that can take place in the context of a View Inheritance pattern, that to the best of our knowledge has not been made explicit so far in ontology modeling. These types of multiple inheritance relationships can be characterized as:

- **Inter-criterion**, when the parent classes involved in the multiple inheritance relation are subclasses of different abstraction criteria. The class C1Class3_C2Class2 in Figure 2 is an example of this type of inheritance because one of its parent classes, C1_Class3, is a refining concept of Criterion1 and the other parent class, C2_Class2, is a refining concept of Criterion2.
- **Intra-criterion**, when the parent classes involved in the multiple inheritance relation are subclasses of the same abstraction criterion. The class C1_Class1Class2 is an example of this type of inheritance because all of its parents classes, C1_Class1 and C1_Class2, are refining concepts of the same criterion, Criterion1.
- **Intra- and inter-criterion**, when there are at least two parents involved in the relation that are subclasses of the same abstraction criterion and there is at least one more different parent that is a subclass of a different abstraction criterion. An example of this type of inheritance is trivial to extrapolate from the composition of the previous two.

[5] Short for: Class_x from Criterion_i
[6] Short for: Class_x from Criterion_i and Class_y from Criterion_j
[7] Short for: Class_x and Class_y from Criterion_i

Nested View Inheritance. It is worth noting that View Inheritance patterns could occur in a nested fashion. That is, a View Inheritance pattern with a concept Ci as root, could enclose another case of a View Inheritance pattern with a different concept Cj as root where Ci subsumes Cj.

As an example, consider the View Inheritance scenario with "Fault" as root in Figure 3. Let us focus in the subtree that originates in one of its abstraction criterion, in this case "BasicViewPointFault" and *zoom in* on it. From the point of view of "BasicViewPointFault" and with this concept as root, its direct subclasses ("ObjectiveFault", "BoundaryFault" and so on) could be regarded as alternative abstraction criteria of "BasicViewPointFault". The classes "FaultType1" and "FaultType2" could be regarded as cases of inter-criterion multiple inheritance given that each one of their parents is a class subsumed by a different abstraction criterion of "BasicViewPointFault".

ODP Classification. According to the classification in [8], the View Inheritance ODP could be considered an Architectural OP given that it provides an ontological characterization of a particular case of polihierarchy in the overall ontology structure. By the same classification, the definition of Inter- and Inter-criterion Multiple Inheritance could be regarded as Logical OPs given that they refine the Logical OP that describes the case of generic multiple inheritance in [7][8]. This classification situates View Inheritance at a higher level of abstraction than that of Inter- and Intra-criterion Multiple Inheritance.

Based on the classification of ODPs in [9], View Inheritance can be regarded as an *extension* to the Normalization ODP, which is considered part of the group of Good Practices ODPs. In this sense, the former concentrates on certain ontological aspects of the modelling problem that the latter addresses. View Inheritance focuses on the nature of a specific type of polyhierarchy structure and provides a characterization of it.

4 Conclusions and Future Work

There are a number of key points that would be worth summarizing. There can be certain ontology domain concepts difficult to represent due to the existing alternative abstraction criteria that define them. That is the case of the "Fault" concept in the context of the ReSIST project.

A survey of the current ontology building techniques was carried out. The Normalization ODP seemed a viable option, yet the pattern did not fully address the definition of "Fault" and the application requirements of the ReSIST project.

To bridge this gap, the View Inheritance ODP is put forward as an extension to the Normalization ODP, combining the latter with the notion of View Inheritance originated in the O-O software design. View Inheritance revealed two basic types of likely relations that could take place in the structure of the pattern: Inter- or Intra-criterion Multiple Inheritance.

These contributions, while not solving all the modelling challenges of the ontology module for ReSIST, do provide additional awareness to be considered in the development process.

Outstanding issues open for future work include the identification of additional examples of real world use cases of View Inheritance. In that sense, we intend to selectively explore the ontologies most frequently used in the data repositories that are part of the Linked Data project[8] for possible occurrences of View Inheritance patterns. The ReSIST project is one of the contributors to the Linked Data set of repositories.

References

1. Gomez-Perez, A., Corcho, O., Fernandez-Lopez, M.: Ontological Engineering : with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web. First Edition (Advanced Information and Knowledge Processing). Springer (July 2004)
2. Gangemi, A.: Ontology design patterns for semantic web content. In Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A., eds.: International Semantic Web Conference. Volume 3729 of Lecture Notes in Computer Science., Springer (2005) 262–276
3. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design patterns: elements of reusable object-oriented software. Addison-Wesley Professional (1995)
4. ReSIST: Resilience and Survivability in IST. Network of Excellence. Contract: IST 4 026764 NOE, EU and Sixth Framework Programme (FP6) and Information Society Technology (IST) (2005–2008) http://www.resist-noe.eu/.
5. Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing **01**(1) (2004) 11–33
6. Rodriguez-Castro, B., Glaser, H.: Whose "fault" is this? untangling domain concepts in ontology design patterns. In: 1st International Workshop on Knowledge Reuse and Reengineering over the Semantic Web at the 5th European Semantic Web Conference. (June 2008)
7. Suarez-Figueroa, M.C., Brockmans, S., Gangemi, A., Gomez-Perez, A., Lehmann, J., Lewen, H., Presutti, V., Sabou, M.: Neon modelling components. NeOn deliverable D5.1.1, Universidad Politecnica de Madrid (2007)
8. Presutti, V., Gangemi, A., David, S., de Cea, G.A., Surez-Figueroa, M.C., Montiel-Ponsoda, E., Poveda, M.: A library of ontology design patterns: reusable solutions for collaborative design of networked ontologies. NeOn deliverable D2.5.1, Institute of Cognitive Sciences and Technologies (CNR) (2008)
9. Egana-Aranguren, M.: Ontology Design Patterns for the Formalisation of Biological Ontologies. MPhil Dissertation, Bio-Health Informatics Group, School of Computer Science, University of Manchester (2005)
10. Rector, A.L.: Modularisation of domain ontologies implemented in description logics and related formalisms including owl. In: K-CAP '03: Proceedings of the 2nd international conference on Knowledge capture, New York, NY, USA, ACM (2003) 121–128
11. Meyer, B.: Object-Oriented Software Construction (Book/CD-ROM) (2nd Edition). Prentice Hall PTR (March 2000)

[8] http://linkeddata.org/