A Software Modeling Approach to Ontology Design via Extensions to ODM and OWL

Rishi Kanth Saripalle, Department of Computer Science & Engineering, University of Connecticut, Storrs, CT, USA

Steven A. Demurjian, Department of Computer Science & Engineering, University of Connecticut, Storrs, CT, USA

Alberto De la Rosa Algarín, Department of Computer Science & Engineering, University of Connecticut, Storrs, CT, USA

Michael Blechner, Department of Pathology, University of Connecticut Health Center, Farmington, CT, USA

ABSTRACT

Ontologies are built to establish standard terminologies representing a semantic agreement between humans and knowledge systems via representational frameworks (e.g., KIF, DAML+OIL, OWL, etc.) that have been proposed in the research community, with limited adoption in industry. One possible reason is a lack of a formal model and associated process to more precisely and accurately design and develop ontologies. The authors' prior work explored UML, entity-relationship diagrams, and XML as compared to RDF and OWL, identifying modeling capabilities lacking in ontologies. In all three approaches, design precedes instantiation which contrasts with ontology developers who build ontologies at the application level targeted to a specific domain. The paper proposes design-level modeling enhancements to ontologies by extending the OMG Ontology Definition Model (ODM) and OWL grammar with capabilities from the three aforementioned approaches, promoting a software engineering-based process. As a result, this work provides a more software engineering-oriented process to ontology design and development.

Keywords: *Ontology Design and Development, Ontology Domain Profile, Ontology Life Cycle Model, Ontology Modeling, Standard Terminologies*

1. INTRODUCTION

Software domain modeling is a process where individuals seek to conceptualize an application in order to arrive at a solution that meets all of the application's domain needs and requirements. Conceptualization can be defined as a structure {D, R}, where D is the domain (scope of the application) and R is the set of relevant relations (functionality and interactions of the application) in the D (Gruber, 2005). Currently, one popular domain conceptualization

DOI: 10.4018/jswis.2013040103

approach is ontologies where the representation of knowledge occurs via the definition of concepts that can be related to one another for sharing and reuse of that knowledge. In practice, W3C supports two major ontology development frameworks, the Resource Description Framework (RDF) (Powers, 2003) and the Web Ontology Language (OWL) (Allemang & Hendler, 2011), where the latter is built on top of the former. Additionally, a wide range of knowledge representational frameworks (e.g., KIF (Genesereth, 1991), KL-ONE (Brachman & Schmolze, 1985), DAML+OIL (Horrocks, 2002), etc.) are available. When reviewing the usage of these frameworks in real applications, the overriding theme is realizing a specific instance-level solution for a particular system, rather than designing a solution that can be reused by multiple similar applications within the same general domain. For example, in health information technology (HIT), multiple systems such as AllScripts (AllScripts, 2012), Centricity (GE Centricity, 2012), MS Health Vault (Microsoft Health Vault., 2013), etc., are developed with their own specific (and different) medical ontologies despite the fact that each one is storing and managing patient data. Any attempt to integrate data from across multiple EHRs, referred to as health information exchange (HIE) requires integrating their different medical ontologies which at best is a semi-automated and arduous task. While many of these HIT systems leverage existing standard medical terminologies (e.g., LONIC (LOINC, 2001), UMLS (Bodenreider, 2004), ICD (ICD, 2009), etc.), there is no uniform attempt to design an ontology domain model for medical knowledge that is general purpose and reusable in multiple contexts. For example, one HIT system may organize medical knowledge in an ontology by disease, symptoms (for each disease), diagnoses (for each disease), and treatments (for each disease), while another HIT system might invert and target this information from a symptom to diagnosis to disease to treatment basis. Attempting to reconcile this medical knowledge in two or more different ontologies is an arborous, time-consuming, and at best semi-automated task, making HIE difficult to achieve.

In order to successfully employ ontologies in existing/new applications, structural and semantic interoperability issues among the ontologies that are used for the overall domain must be addressed. There are a number of key issues. First, the individual ontologies of each constituent system used by a new application may each organize knowledge in different ways to suit their specific application and organizational processes, meaning that the ontologies across the constituent systems are often incompatible and difficult to integrate. Second, the ontology development and deployment process is predominantly instance and construction based, often dictated by the talent and expertise of the ontology designer rather than using any concrete software development process; such an approach limits the reuse since ontologies end up being very domain centric. For a new application, the existence of consistent ontologies of the constituent systems will greatly simplify the semantic interoperability. Finally, many existing ontology representational frameworks lack an ability to design solutions that are broader in scope; the end result is often narrowed to not just a single domain, but to a subset of the domain that is very application specific. Thus, the overriding issue is that ontologies solely focus on the domain knowledge and its usage by constituent systems rather than abstracting back from the problem to consider the entire domain and its appropriate set of ontologies in a more comprehensive and general manner. Clearly, there is a lack of design and process in the current ontology definition process – focusing solely on the domain knowledge and its usage by a particular application rather than abstracting back from the problem to consider domain knowledge in a more comprehensive and general manner (Kuhn, 2010). Such an approach is contrary to the long history of design in software, databases, and web settings, where the emphasis is on individual modeling techniques that can applied to conceptualize problem solutions in a fashion that promotes abstraction and fosters reuse. In computing, there is a wide

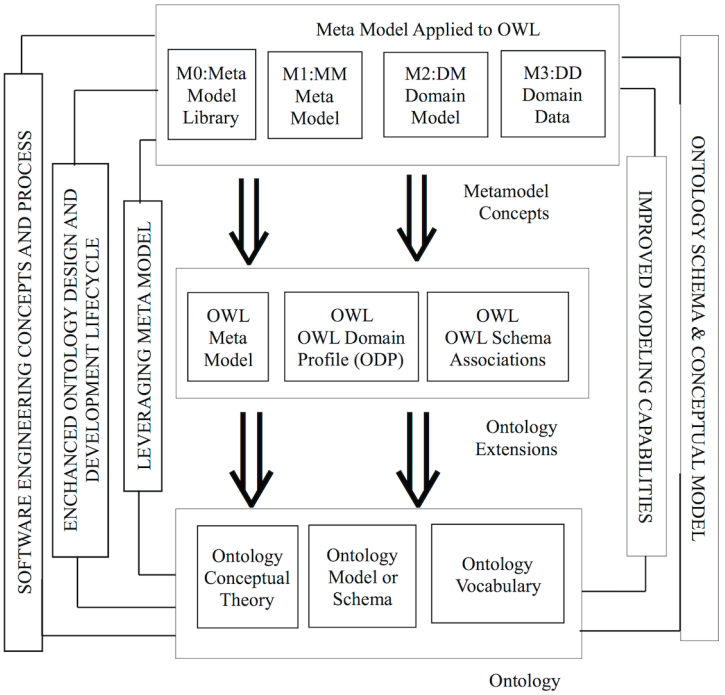
range of such modeling techniques ranging from programming (e.g., C#/C++/Java, LISP, Prolog, etc.) to software design (e.g., the unified modeling language (UML) (Booch, Rumbaigh, & Jacobson, 2005), design patters, software architectures, etc.) to databases (e.g., entity-relationship diagrams (ERD) (Chen, 1976), relational theory and normalization, etc.) to the world wide web (e.g., the eXtensible Markup Language (XML) (Harold & Scott, 2004) and associated technologies). All of these modeling techniques have underling conceptual models (e.g., object-oriented, functional, formal logic, relations, etc.), a means to access the conceptual model (e.g., programming language, SQL, XSLT, etc.), and in some situations, the existence of a metamodel that allows for the model itself to be modified and changed (e.g., the UML metamodel, meta-programming through annotations in Java, etc.).

To illustrate these concepts and their associated processes, consider UML, ERD, and XML. UML is a metamodel driven approach that provides diagrammatic models (e.g., use case diagram, class diagram, sequence, diagram, activity diagram etc.) and an extension mechanism for extending UML modeling entities (like class, association, property etc.) to build domain specific metamodel elements. In database design, ERD focus on abstract entities, relationships between entities, and other capabilities that can be transitioned to formal relational database schema; the schema describes the tables, dependencies, keys, referential integrity, etc. XML has emerged as a de facto standard for modeling and data exchange and has revolutionized the way information is represented, transmitted, and shared across heterogeneous systems. XML is used in a wide variety of settings (e.g., document representation, database exchange, medical standards such as HL7, access control (XACML), security, etc.), but is still focused on the modeling of data rather than capturing information and knowledge (the meaning or semantics of data/information). Due to the lack of standard semantic interpretation of information represented in XML, there has been a movement by researchers to augment data with knowledge, as evidenced by the development of the aforementioned knowledge representational frameworks supported by RDF and OWL to encode data, with a major focus on the use and development of ontologies. This approach juxtaposes UML, ERD, and XML, where the emphasis is on concentrating on the design level considerations and defers the implementation. In emphasizing modeling, one of our objectives of this paper is to enhance knowledge representational languages with modeling capabilities to improve design and usability.

The overall intent of our research, as shown in Figure 1, is to develop a framework for ontology modeling, design, and development to represent a process that spans the entire gamut from the metamodel to the realization of an actual ontology, moving the process from one that is instance-based and domain specific to align more closely with a software engineering process and a focus on modeling and design. The core of the work proposed in this paper is captured in the horizontal boxes *Metamodel Concepts*, *Ontology Extensions*, and *Ontology* in Figure 1.

For the *Metamodel Concepts*, we leveraged the MOF Metal Model Library (M0) in conjunction with the metamodel (M1) concepts of UML, ERD and XML as applied to OWL allowing us to propose Ontology Extensions to OWL allowing OWL concepts to be aligned to the UML metamodel, an OWL Domain Profile (ODP) that is used to capture domain generic concepts at a metamodel level, and OWL Schema Associations that allow high level associations between OWL schemas. The last step, the Ontology box in Figure 1, utilizes *Metamodel Concepts* and *Ontology Extensions* to design an ontology that is supported by an underlying conceptual model defined with one or more models and/or schemas using a given ontology vocabulary, resulting in an instance of the ontology. The vertical boxes in Figure 1 clearly illustrates the interactions between different components (horizontal boxes) of the proposed framework. The Software Engineering Concepts and Process (far left of Figure 1) spans

Figure 1. A complete framework for ontology design and development



from Metamodel Concepts to the realization of the Ontology; this is the software design process aspects of our work in the framework that is achieved by proposing Hybrid Ontology Design and Development with Life Cycle (HOD²LC) (second left box, Figure 1), an agile enhanced ontology design and development lifecycle for developing an Ontology (bottom middle box) utilizing software concepts spanning the horizontal boxes.

In this paper, the main goal is to elevate ontology models and their design and development process so that the end result aligns more closely to traditional modeling processes that are widespread in software engineering. Towards this goal, we utilize our previous work (Saripalle, Demurjian, & Behre, 2011) that contrasted the features of OWL against UML, ERD, and XML and proposed conceptual extensions to OWL from varied perspectives in terms of attributes, profiles, and schema associations. Our prior work focused on conceptual extensions to OWL to support a more software-engineering based process for constructing ontologies; this paper extends this work with specific model recommendations to more fully leverage OWL modeling capabilities by adding design-level modeling concepts and capabilities, and most importantly, propose extensions to Ontology Definition Model (ODM) (ODM, 2009) and OWL to incorporate these recommendations. We transition the design capabilities of UML, ERD, and XML to identify new capabilities to be included in ODM and OWL that augment ontology models and frameworks to upgrade the usage of ontologies to an engineering process.

The remainder of this paper has seven sections. Section 2 presents a realistic health care scenario based on EHR's, their supporting ontology knowledge concepts, and the knowledge required for the queries the clinical researchers may be interested in making across multiple HIT systems. Section 3 briefly summarizes our previous work (Saripalle, Demurjian, & Behre, 2011) that compared and contrasted UML, ERD, and XML to OWL. Using this as a basis,

Section 4 explores concepts related to domain modeling by applying UML's layered OMG's Meta Object Facility (MOF) to XML, RDF, and OWL (Meta Model in Figure 1); as a result, OWL is aligned at a higher conceptual design level away from the more traditional instanced-based approach to more easily support our proposed ontology extensions. Section 5 proposes and explains the ontology extensions (see Figure 1) of Attribute, Domain Profile, and Schema Associations to OWL and ODM in combination with the Ontology Metadata Vocabulary (OMV) framework. Section 6 examines a Hybrid Ontology Design and Development Life Cycle (HOD²LC) (Saripalle & Demurjian, 2012) for ontology design, development, and deployment. Section 7 reviews related research with Section 8 offering concluding remarks and ongoing research.

2. A HEALTH CARE SCENARIO

This section presents a clinical scenario of health care based on HIT systems, their supporting ontologies, and the knowledge required for the types of queries the clinical researchers may be interested in making across multiple systems. The intent is to provide a realistic scenario of patient care that includes: a patient's relevant medical problem, involved laboratory tests and results, the resulting medical diagnosis, the role of involved HIT systems in the process, motivation to fully place our work in this paper in an appropriate context, and the potential usage of the information by providers and clinical researchers. Note that this example was formulated by co-author M. Blechner, MD who has used his medical expertise to provide a real-world scenario and its usage in practice:

- **Patient History and Initial Findings:** A 72 year old male, Mr. Smith, with a history of type 2 diabetes presents to the emergency room complaining of shortness of breath (also known as dyspnea) on exertion. He reports experiencing increased difficulty climbing the one flight of stairs in his house. Mr. Smith also indicates experiencing occasional chest pressure on exertion (stable angina). He has recently developed swelling in his ankles and feet (edema). He indicates that he takes metformin (for blood glucose control) for his diabetes and benazepril for his blood pressure, although he does not recall the doses. He also takes an aspirin a day because his regular doctor told him that he should. The physical exam of Mr. Smith reveals a gentleman in mild respiratory distress with moderate pedal (foot) and lower extremity edema (fluid in tissues). He is tachypneic (increased respiratory rate) with a respiratory rate of 30/min. Chest auscultation (listening to the lungs with a stethoscope while the patient breathes) reveals bilateral basilar rales (crackling sounds at the base of both lungs) and the neck shows jugular venous distension (a visible distension of the jugular vein in the neck that typically represents backup of blood returning to the heart which is typically due to congestive heart failure - CHF). He has a regular pulse at 90 beats per minute and blood pressure of 140/90 (mildly elevated). Oxygen saturation (a measurement of the amount of oxygen being carried in the blood, normally around 98%) on room air (what your breathing now) is 88% and rises to 98% when given supplemental oxygen by nasal cannula at 2 liters/min (we supplement the air the patient is breathing through the nose). An electrocardiogram (EKG) measures the electrical activity in the heart that is responsible for heart pumping) shows a normal rate and rhythm with evidence of left ventricular hypertrophy (the muscle of the left side of the heart is thickened) but no ischemia (i.e., at the time the EKG was being performed there was no evidence of inadequate blood flow to the heart muscle);
- **Laboratory Tests and Results:** The ER physician as part of the evaluation process ordered a series of laboratory blood tests ordered to be performed to assist in the

evaluation of Mr. Smith. Specifically, the laboratory tests ordered on Mr. Smith were (B-natriuretic peptide (BNP), Hemoglobin, Hematocrit, white blood count (WBC), sodium (NA), potassium (K), chloride (Cl), bicarbonate (CO2), glucose, blood urea nitrogen (BUN), creatinine (Cr), and Troponin I) and their results as impacting Mr. Smith yield as given in Table 1.

To explain and review the results, BNP (B-natriuretic peptide) is a compound released by the heart when it is stretched. In CHF, blood returning to the heart is not pumped out quickly enough and as result the heart stretches to accommodate the extra blood. Thus, the elevated BNP is very suggestive of CHF. Hemoglobin and hematocrit are both measurements of the amount of red blood cells in the patient's circulation. Na (sodium), K (potassium), Cl (chloride), CO2 (bicarbonate) are all measurements of ion/electrolyte concentrations in the patient's blood. In the case of Mr. Smith, while these results are in the normal range, CHF can often result in imbalances to these electrolytes. GLUC is the patient's serum glucose and the elevation in the test result in this case may indicate poor control of his diabetes. However, CHF is a stressful condition and serum glucose values typically rise during physiologic stress even in non-diabetics and otherwise healthy patients so this may not represent poor glucose control. BUN (blood urea nitrogen) and Cr (creatinine) are typically used to assess kidney function. The elevated BUN and normal creatinine of Mr. Smith suggest that the BUN is elevated due to decreased blood flow through the kidneys rather than due to a problem with the kidneys themselves. This increased BUN can be referred to as azotemia and since it is due to poor kidney perfusion (a variable that has its effect before the kidney); it is often referred to as pre-renal azotemia. The decreased kidney perfusion in Mr. Smith is due to the CHF in this case. Troponin I is a protein found only in heart muscle cells. Elevations in troponin I indicate damage to heart muscle cells which spill the protein into the blood. The normal troponin I in Mr. Smith's case indicates that there has not been a heart attack and is consistent with the lack of evidence of ischemia seen on EKG. Overall, the Lab tests for Mr. Smith reveal a marked elevation in the BNP suggestive of congestive heart failure. The elevated BUN and normal creatinine suggest a possible pre-renal azotemia which would also be consistent with CHF. The normal troponin is consistent with the lack of evidence of ischemia seen on EKG:

- **Medical Diagnosis based on Examination, EKG, and Labs:** Mr. Smith is suffering from an exacerbation of CHF. The pumping function of his heart is not powerful enough to move all of his blood volume through his circulation. As a result, the blood that is returning to the heart backs-up and essentially pools in the blood vessels. In the legs and feet, gravity tends to accentuate this pooling. The extra pooled blood in these parts of the circulation exerts increased pressure on the vessel walls and this increased pressure pushes fluid from the circulation into the body tissues. This fluid and the resulting puffiness of the tis-

Table 1. Lab test ranges and value results

Test (Ranges)	Value	Test (Ranges)	Value	Test (Ranges)	Value	Test (Ranges)	Value
BNP (<100 pg/mL)	800 pg/mL	Hemoglobin (13.5 – 17.5 g/dL)	13.5 g/dl	Hematocrit (42 -52%)	41%	WBC (4 – 10 k/uL)	8 k/ul
Na (135 - 145 meq/L)	142 meq/L	K (3.5 – 5.5 meq/L)	4.3 meq/L	CI (100 – 111 meq/L)	99 meq/L	CO2 (23 –32 meq/L)	31 meq/L
GLUC (<200 pg/mL)	250 mg/L	BUN (8 -24 mg/dL)	33 ml/dL	Cr (0.6– 1.2 meq/L)	1.1 mg/L	Troponin I (<0.05 ng/mL)	<0.05 ng/mL

sues are called edema. In CHF, edema fluid also collects in the lungs and this fluid in the lung impairs the lungs ability to absorb oxygen, thus the difficulty breathing and the increased respiratory rate as an attempt to compensate. The rales (crackles) heard on auscultation are an indication of fluid in the lungs. So both oxygenation of the blood (due to fluid in lungs or "pulmonary edema") and delivery of blood to tissues (the heart's pumping function is inadequate) is compromised. Together, these two factors compromise oxygen delivery to the tissues resulting in fatigue (thus, the difficulty climbing stairs) and potentially organ dysfunction. The jugular vein returns blood from the head and brain to the heart. The jugular venous distension is another manifestation of pooling of blood due to inadequate cardiac function. The occasional chest pressure on exertion suggests that Mr. Smith has coronary artery disease where inflammation and deposition of cholesterol in the vessel wall results in a localized expansion of the vessel wall (typically referred to as a plaque) that can impede blood flow. When the patient is at rest, there is adequate blood flow past this plaque but when the patient exerts themselves, the heart needs to work harder and needs more oxygen and thus more blood flow. If the plaque is large enough, blood flow becomes inadequate on exertion and heart muscle can become damaged. This can be felt as chest pressure or pain. If Mr. Smith stops exertion, the heart stops working so hard, the oxygen demand drops, oxygen delivery is once again adequate and the chest pressure goes away. This transient chest pain or discomfort is called stable angina. If the blood flow is too compromised for too long, heart muscle dies of a "heart attack" or myocardial infarction. Mr. Smith has stable angina and no evidence of a myocardial infarction. Mr. Smith's blood pressure is slightly elevated but since he has been prescribed antihypertensive medication (benazepril), this suggests that the medication or dose may need to be changed or that the patient is not following the prescription. Diabetics are often hypertensive and hypertension leads to left ventricular hypertrophy and eventually CHF as well as coronary artery disease. Aspirin decreases the risk of heart attack and is often recommended in patients with cardiovascular disease like him;

- **Role of HIT and Discharge:** Mr. Smith provided the ER physician with access to his personal health record (PHR) that had been recently initiated by the patient's son who was not present. The PHR data was sparse but did include that the patient had been taking flax seed oil supplements for the past 6 months. A search of the regional HIE revealed that the patient had a recent admission at another hospital for CHF; thus data must be gathered from that electronic health record (HER), his primary physicians EHR, and potentially others in support of his ongoing care. The discharge summary from that admission indicated that the patient had improved after 2 doses of Lasix (diuretic, makes you pee a lot and the loss of fluid decreases your blood volume which often improves CHF symptoms) and the patient had been discharged on Lasix for use at home. A query of the Superscripts electronic prescription database, however, suggested that the patient never filled the Lasix prescription. The HIE also informed the ER physician that the patient has a documented allergy to sulfa containing medications;
- **Motivation of Scenario as Related to Paper:** In practice, information from multiple EHRs are brought together via HIE in order to allow data not only to be shared, but for physician researchers to be able to make queries across repositories on various medical concepts. We assume a situation in the not so far future that each of these EHRs has their own ontology to represent product-specific information within the EHR, and utilizes the various standard medical ontologies (e.g., LOINC,

UMLS, customized ontology, etc.) to store information in either CCD or CCR formats, or in a proprietary data format. The ontologies for each EHR must be integrated into a global ontology in order for the data to be effectively shared so that differences in ontologies can be reconciled. In this case, a global ontology is created by utilizing information from existing ontologies. For example, EHR-specific ontologies may contain information on diagnosis (the disease or condition that a patient has), symptoms (the discomfort or issue that a patient is having), treatments (the regime, medication, or therapy order for the disease), etc. The two key issues with such product specific ontologies are two-fold. First, each of the EHRs may organize the information hierarchically into ontologies in different ways; one may use symptom as the highest level to reference diseases which in turn references treatments, while another may use disease to organize the symptoms and their treatments. The problem is that if one tries the search the first EHR for all treatments for the different diseases that have the same symptoms, the complexity of the search is difficult since it is structured differently than the ontology, and the same search is different for the second EHR that has an alternative organization for the same ontology Second, the same or similar terms in the ontologies themselves may be semantically interpreted in alternative ways by different EHRs; cardiac failure in one ontology and heart attack in another ontology. This is why there needs to be syntactic and semantic unification to reconcile and integrate ontologies. One objective with our work is to define a global ontology using our extended ontology model and HOD2MLC in order to allow this integration based on the integration of terms, structure, and semantics by alternate ontologies from different EHRs. A complementary approach would start with the creation of a global meta-ontology from existing information on ontologies that currently exist in the public domain, resulting in a global meta-ontology that can be translated to platform specific ontologies. While our research focuses on the former, our long-term goal is to achieve the later, with the potential to semi or full automate the translation between meta-ontologies and local ontologies;

- **Potential Usage of Scenario in Clinical Practice and for Research:** The scenario as presented above has the underlying aspect of information that has been gathered for this patient from multiple data sources that have been utilized for his treatment. If one extrapolates across all of the patients that may have similar symptomatology, there is the opportunity for improved clinical practice or utilization by a physician researcher who is interested in conducting research on patients that have a profile similar to Mr. Smith (as outline in the scenario). The intent is to transition from using the data for clinical purposes to leveraging the data for research. We want to model and define a set of underlying ontologies based on the clinical data from the scenario that can then facilitate research across a broader spectrum. From a research perspective, an almost limitless number of questions can arise when considering Mr. Smith's case or similar patients. This common clinical scenario is associated with a number of clinical variables from the presence or absence and magnitude of various symptoms or physical exam findings such as dyspnea (shortness of breath) or edema (swelling) to the administration and dose of various medications, to the results of various laboratory tests to the patient's clinical diagnoses.

Correlations between two or more of any of these variables may be of interest to a researcher. A number of broad research topics (RT) can be posed:

RT.1: Effects of a specific medical therapy on a patient's co-morbid conditions:

1. How does metformin used for glucose control in type 2 diabetics effect the incidence and natural history of CHF and chronic renal (kidney) Failure or stable angina (chest pain successfully treated)?
2. If there is an effect on any of these conditions, is type 2 diabetics dose dependent on metformin or does the absence of any other medications alter it?
3. Does metformin affect the utility of the BNP test (measure of how well the heart is working) for the diagnosis and monitoring of CHF?
4. Does metformin decrease the risk of developing any seemingly unrelated co-morbid diseases like breast cancer?

Many of these questions would be answerable if you had a large enough integrated database supported by ontologies that would be able to extract all of the required data and attempt to identify for markers of CHF across both long-term and short-term periods of time:

RT.2: Comparative study of different diabetic therapies with CHF using various patient groups:

1. What is the patient's profile with CHF and associated medications involved for diabetic therapies?
2. What are incidence and/or severity of CHF for diabetes patients who use hyperglecimic agents?
3. Is metformin more or less effective in maintaining glucose control in type 2 diabetics with a history of stable angina as compared to other anti-hyperglycemic medications?
4. Is lasix (reduces edema) more or less effective than alternative diuretics in treating CHF in patients with type 2 diabetes?

RT.3: Biomarkers (measureable characteristics) of disease, disease progression or risk:

1. Are there patterns of laboratory test results seen in type 2 diabetes patients that are associated with increased risk of developing CHF or Stable Angina?
2. If so, do any specific medical therapies alter this risk?

RT.4: Adverse events associated with specific medical therapies:

1. Adverse events associated with a specific drug like metformin may not be detectible in the entire metformin treated population but may be significant in a specific subpopulation of patients. Are there any subpopulations of type 2 diabetics on metformin that reveal a significant adverse event rate? Diabetics with CHF? Diabetics with renal failure? Diabetics also treated with lasix for CHF?
2. Is there a subpopulation of patients taking flax seed oil supplements that reveals a significant incidence of adverse events? Perhaps diabetics on metformin with concurrent CHF?

Individually, each of these research topics provides a means to allow a physician researcher to explore various aspects of a disease, its symptoms, medications, therapies, interactions with other conditions, etc. To accomplish this, it will be necessary not only to integrate the data sources, but to provide a consistent set of ontologies that describe the information in a manner that will allow queries from differing perspectives to be posed and answered. The role of our research on ontology design is in support of such a process.

3. PRIOR WORK ON MODELS AND ONTOLOGIES

This section briefly reviews our prior work (Saripalle, Demurjian, & Behre, 2011) comparing UML, ERD, and XML to OWL at a class and instance level. UML provides diagrammatic models (e.g., use case, class, sequence, etc.) with the focus on artifacts (classes/types)

rather than instances (objects). ERD supports database requirements via entities to model aggregations (set of attributes) and relationships to model static associations between entities with cardinalities (e.g., one-to-one, one-to-many, etc.) and inheritance. XML supports information content to be hierarchically organized and tagged which can be defined and validated by XML Schema Definition (XSD) achieving type-level characteristics that are enforced by all type instances. RDF captures knowledge with the RDF Schema (RDFS) framework supporting classes, properties, and restrictions. OWL extends the RDF/RDFS to improve expressiveness in terms of qualified restriction, number restriction, unique keys, boolean expression, etc. From a modeling perspective, UML, ERD, and XML share: *abstraction* to hide implementation details and *aggregation* to group attributes or properties into a named concept, namely, an entity in ERD, a class in UML, a table in a relational database schema, etc.; *schema definition* for a conceptual model that describes system structure and behavior; *schema association* to allow relationships among the logical schemas; *classes* (types) to aggregate objects that share common attributes; *attribute* (properties) that are characteristics which are owned by the class; *interface* that abstractly defines the behavioral aspects (operations) of the implementing class; *associations* to relate two or more classes (types or entities, etc.); *inheritance* for extension (child is enhanced), *specialization* (child is restricted), *generalization* (common attributes of classes are abstracted to form parent), and *combination* (inherit from multiple classes among classes (types); and, *constraints* to limit information in schemas. *Associations* can be qualified (based on a value), at the class level, n-array between multiple classes, and with a limited number of instances (cardinality).

In Table 2, we compare UML, ERD, and XML using three qualitative criteria: *None*, the model does not support the feature; *Partial*, the model has some aspects of the feature; and, *Full*, the model has all aspects of the feature. The comparisons of UML, ERD, and XML are quite clear given the earlier discussion.

Table 2. Model characteristics vs. UML, ERD, XML and OWL

Modeling Element	UML	ERD	XML	OWL
Schema Definition	Full	None	Full	**None**
Schema Associations	Full	None	Partial	**Partial**
Interfaces	Full	None	None	**None**
Class	Full	Full	Full	**Partial**
Associations	Full	Full	Full	**Partial**
Qualified Associations	Full	Full	Full	Partial
Association Class	Full	Full	Full	None
N-Array Associations	Full	Full	Full	Full
Cardinality	Full	Full	Full	Full
Inheritance	Full	Full	Full	Full
Extension	Full	Full	Full	Full
Specialization	Full	Full	Full	Full
Generalization	Full	Full	Full	Full
Combination	Full	Full	Full	Full
Constraints	Full	Full	Full	Full
Profile	Full	None	None	None

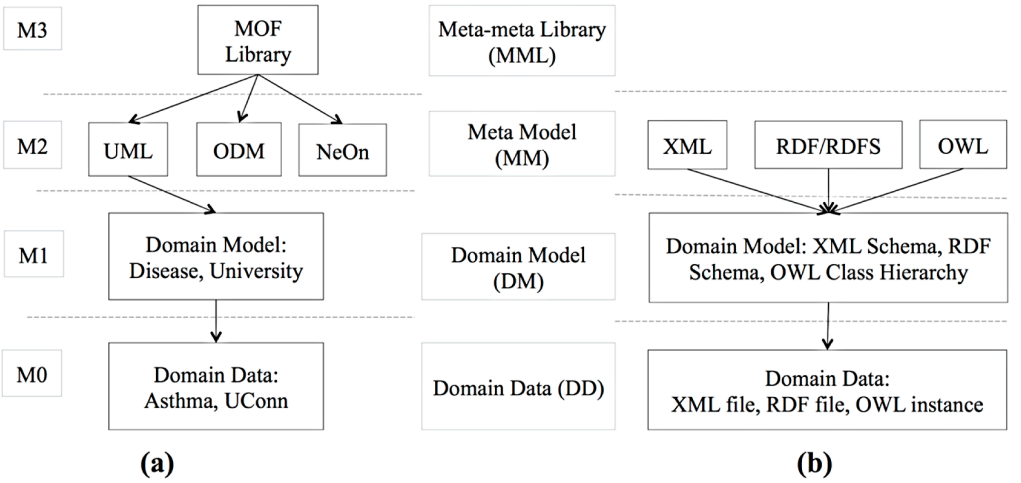
What is more relevant is an evaluation of the features vs. ontologies/OWL. While schema definition in OWL is not directly supported, associations of schemas are possible to a limited extent. For classes and interfaces in OWL, owl:Class is defined as a set of individuals and owl:ObjectProperty links two individuals, but the link can't differentiate between a relationship and an attribute and the definition of the class in OWL and modeling are semantically different. As a result, there is a lack of support in OWL for schema definition. Due to the lack of the semantic definition of a class and the concept attribute, developers can't capture the structure of the concepts and ontology itself, resulting in the "*None*" for schema in Table 2. Presently, ontologies can refer to multiple ontologies, but can't define customized associations between them, thus, Partial support. The Associations and Inheritance are present in OWL grammar, are designed to be used with OWL class (in the context of the OWL).

4. A LAYERED MOF ARCHITECTURAL APPROACH

This section applies UML's layered OMG's Meta Object Facility (MOF) (OMG, 2011) to XML, RFD, and OWL to represent domain modeling as shown in the metamodel box in Figure 1, to allow designers to understand the function and capabilities of the planned system by conceptualizing an abstract view of the solution. For this paper, we are interested in not only the modeling techniques, but also the underlying metamodels of these techniques that provide the means to specify the model, its properties, and its semantics. This section explores domain modeling of XML, RDF, and OWL via a layered MOF approach, focusing not only on their individual modeling capabilities, but on also understanding their underlying metamodels and the way that these metamodels can be exploited to add a more software engineering emphasis to ontology design and development. The intent is to transition from an instance-based process for ontology creation to one that is design based and has an underlying formalism (the metamodel) that elevates the process to one that is at a higher conceptual level.

As a model, UML provides diagrams (e.g., class, use case, etc.) at varied levels of abstraction to represent associations among different concepts (e.g., actors, use case, class, etc.). Underlying these diagrams and modeling concepts is the UML metamodel which is built on top of MOF as given in Figure 2a: M3 is a meta-meta library for defining new metamodels; M2 is at the metamodel layer where models such as UML, ODM, and NeOn (Haase, Rudolph, Wang, & Brockmans, 2005) can be defined; M1 is the domain model instance of an M2 metamodel; and, M0 is the instance of the domain model. Building a metamodel using MOF for specific domain requirements is a tedious task, as the developer has to define the syntax and the semantics of the new entities. To facilitate this process, it is possible to utilize the UML abstraction *Profile* (Fuentes-Fernández & Vallecillo-Moreno, 2004), where existing UML metamodel elements such as *class, association, property,* etc., can be extended to build domain specific metamodel elements. UML *profile* only extends existing UML metamodel features, allowing the profile extension to act as a metamodel itself. In addition to M3 and M2, Figure 2a also shows the transition from the metamodel UML (M2) to a domain model for a university application (M1) applied to a "real" university UConn (M0). Likewise, an ODM, metamodel (M2) can be used to develop a domain model (M1) for Diseases that can then be applied at the domain data layer (M0) for various diseases (e.g., Asthma). Three of the four layers of MOF in Figure 2a (metamodel, domain model, and domain data) can be used to organize a conceptual view of XML Schema, RDF/RDFS schema, and OWL metamodel, as shown in Figure 2b. In the case of XML, the capability of XML Schema Definition (XSD) at the MM layer (XML schema for short) provides predefined schema tags like *element*, *complexType*, *simpleType,* etc. These schema tags are akin to classes, attributes, actors, etc.,

Figure 2. A layered organization of metamodels



in UML. Through this approach, the definition of an XML schema at the domain model (DM) level can be transitioned to an instance level (domain data – DD).

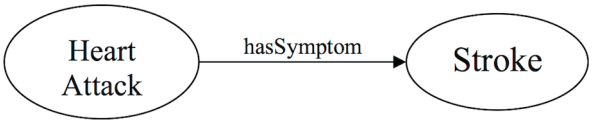
XML has the flexibility in representing the information, but lacks support in providing semantic interpretation/meaning to the information represented. For example, consider a sample XML snippet in Figure 3, humans can conclude that the concept *"Stroke"* is a symptom of the concept *"Heart Attack"*, but a knowledge-based system could also interpret the information as *"Heart Attack"* has a symptom of *"Stroke"* (Figure 4). XML provides a flexible framework for representing information at the schema level (domain model) and its corresponding instances (domain data), but the semantics of the structure have to be mutually agreed on between the exchanging systems in order for data to be successfully shared.

The semantic ambiguity in interpreting XML elements has led to RDF (MM in Figure 2b) which leverages the structure of XML by annotating data and its structure with semantics. RDF has tags that have a similar syntax to XML (DM in Figure 2b) and utilizes the data types defined in the XML Schema grammar. For example, in Figure 4, Heart Attack – hasSymptom – Stroke is an RDF triple statement where "Heart Attack" is the *Subject*, "hasSymptom" is the *Predicate* and "Stroke" is the *Object*. This supports a transition from the domain model to the domain data layer. RDFS (MM in Figure 2b) was developed for providing schema modeling elements like class (rdfs:Class) and property elements like rdfs:subClassof, rdfs:subPropertyOf, etc. However, both RDF and RDFS are constrained with respect to the expressiveness (i.e., axioms, unique key, and reasoning, etc.) to build complex knowledge

Figure 3. XML example illustrating the semantic ambiguity in data interpretation

```
<xs:complexType name="hasSymptom">
    <xs:sequence>
      <xs:element  ref="Heart Attack"/>
      <xs:element  ref="Stroke"/>
    </xs:sequence>
<xs:complexType />
```

Figure 4. RDF statement example in triple format



structures. OWL (OWL Guide, 2004), also at M2 in Figure 2b, is built on top of RDF/RDFS to take advantage of RDF triples and provides richer semantics with greater expressive power for defining complex ontologies via three variants: *OWL Lite* an easy to use functional subset of OWL; *OWL Description Logic (DL)* to support the existing description logic and to provide a language subset that has desirable computational properties for reasoning systems; and, *OWL Full* that relaxes some of the constraints on OWL DL. However, OWL Full semantics are undecidable (Motik, 2005) making OWL DL the most popular framework used for developing ontologies that combine *expressiveness* and *complexity*; as a result, our focus is on OWL DL.

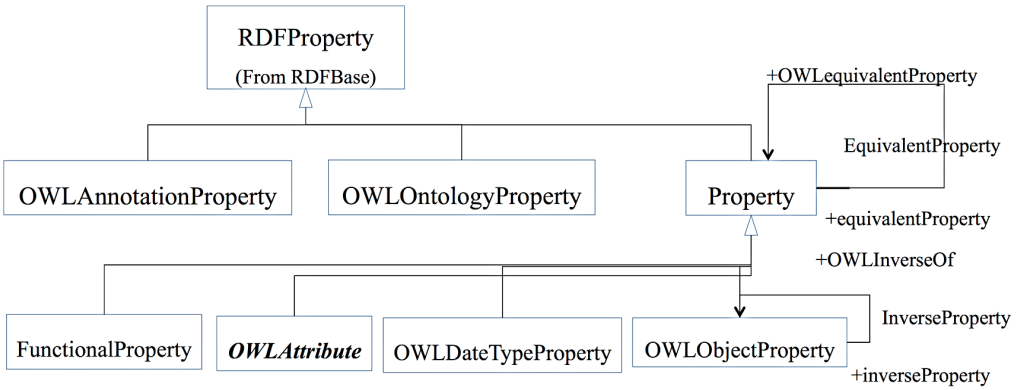
Concepts in OWL DL are represented as classes of type owl:Class, which can be instantiated to form data instances of type owl:NamedIndividual. OWL DL has the uniqueness to define new classes from existing classes via boolean operators such as *union, intersection, negation,* and *number restrictions*. OWL DL specifications also have three role variants, as shown in Figure 5, which act as "*binary relationships* or *associations*" between concepts or classes of type owl:Class (*hasSymptom* in Figure 4), defined as:

1. **ObjectProperty:** Is a binary relation or a role between two individuals:

<owl:ObjectProperty
rdf:ID="hasSymptom">
"hasSymptom" relates two individuals as shown in Figure4

2. **DateTypeProperty:** Represents the properties of classes whose values are datatype like Integer, URI, and String, etc. The OWL grammar uses most of the built-in XML Schema datatypes:

Figure 5. An extended OWL property metamodel in ODM



<owl:DatatypeProperty
rdf:ID=”hasScienticName”>

3. **AnnotationProperty:** Allows annotation of various ontology concepts using rdfs:Comment, rdfs:SeeAlso, etc. In ODM, OWLAnnotationProperty entity represents <owl:AnnotationProperty> as shown in Figure 5.

In summary, this section provides the reader with a basic understanding of RDF/RDFS and OWL frameworks and their potential alignment with MOF layers. The OWL Semantic Web guide (Lacy, 2005; Motik, 2005; Motik, Patel-Schneider, & Grau, 2009) provides in-depth knowledge on the model-theoretic semantics of the framework and the work of (Horrocks, Sattler, & Tobies, 1999) explains the semantics in detail.

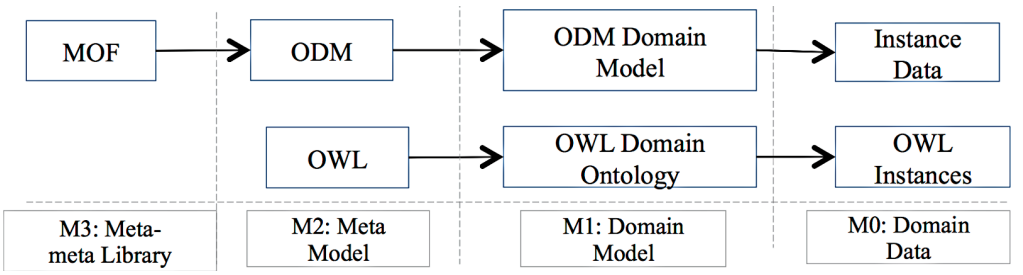
5. EXTENSIONS TO OWL, ODM, AND OMV

Ontologies are primarily built to support knowledge-based systems or the key terms and their relationships for a given domain, where an ontology designer focuses on defining a specific instance-based solution for the given application. As a result, the ability to reuse and share of existing knowledge to facilitate semantic interoperability between information exchanging systems is difficult to achieve when ontologies are constructed for such specific and narrow purposes. OWL provides a framework for embedding knowledge semantics and to develop complex ontologies. Section 3 identified the key capabilities missing in OWL when compared to UML, ERD, and XML, resulting in a lack of software modeling concepts and an engineering process when designing, developing, and deploying ontologies. These missing components in OWL make the ontology integration process to be highly inefficient and laborious in nature; the emphasis has been on building an ontology instance-based solution which limits the ability to reuse the ontology in other applications of the same domain.

The objective of this section is to propose three extensions to the OWL and ODM frameworks in combination with the Ontology Metadata Vocabulary (OMV) (Hartmann, Palma, & Sure, 2005) to yield an approach that improves the design and modeling capabilities thereby defining a software engineering process for ontology design, development, and deployment (see Section 6). As shown in Figure 6, we leverage MOF concepts and the four layers (M3 to M0) and the discussion in Section 3 in order to transition to an engineering-based process for ontology construction. OWL provides a framework for developing ontologies and enables reasoning by exploiting the underlying description logic representation of the knowledge, shown in the middle of Figure 6.

ODM is an *instance* (indicated by solid black arrow) of MOF as shown in Figure 6, and provides visual modeling diagrams for developing ontologies. As a result, this section provides a detailed discussion on two recommendations for extending OWL, ODM, and OMV in order

Figure 6. Representing OWL and ODM at the various design phases

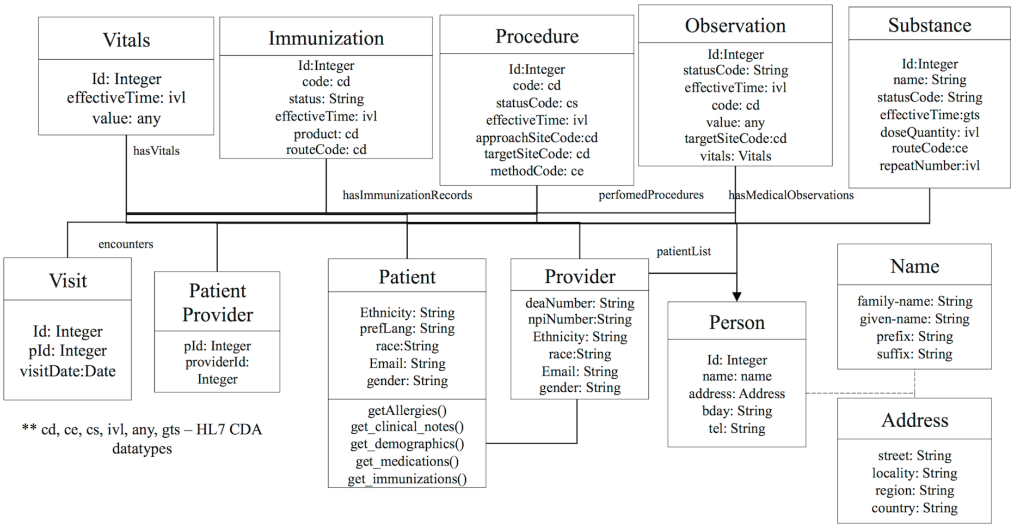


to enhance modeling capabilities of ontology models. The remainder of this section details our proposed OWL extensions for the framework of Figure 1. Section 5.1 examines the first proposed extension to OWL that adds *Attribute* as a means capture the characteristics of a class. Section 5.2 details the second proposed extension to OWL that adds *Domain Profile* to OWL in order to capture domain specific concepts at the meta-model (M2 in Figure 6) level. Finally, Section 5.3 presents the third extension for schema associations by leveraging OMV to define *Ontology Schema Associations* to work with the two extensions in Sections 5.1 and 5.2 to define associations. The first two extensions leverage the modeling capabilities of the OWL/ODM meta-model, thereby influencing the OWL domain model design; the usage of OMV's to define ontology associations will allow dependencies across ontologies to be defined. Note that further implementation details of the work in this paper that integrate our OWL extensions into the Protégé tool (Protege, 2012) and provide an implementation of the ontology in this paper are provided at the web site: http://www.engr.uconn.edu/~steve/KanthSaripalle.html.

5.1. The Owl Attribute Extension

This section examines the first extension to OWL that adds *Attribute* as a means capture the characteristics of a class as defined in Section 4; this is akin to attributes in UML meta-model (Booch, Rumbaigh, & Jacobson, 2005). The reason that we are proposing attributes is to augment OWL with the capability to capture characteristics owned by a class and define an OWL *class* as an aggregation of attributes. When the semantics of OWL DL are utilized for this purpose, the result is that a class (identified as a *concept* in DL) is formed by grouping a set of objects (Horrocks, Sattler, & Tobies, 1999; Kuhn, 2010), but not by identifying and grouping the attributes of those objects (Baader, McGuinness, Nardi, & Patel-Schneider, 2005). To further support this argument, consider the UML diagram shown in the Figure 7, which is a subset of the XML standard Health Language Seven (HL7) Standard Clinical Document Architecture (CDA) (Boone, 2011). The *Patient* class has *attributes id* (type Integer*), Weight* (type Double), *Height* (type Double), and *hasName* (type Name), and is associated with the *Provider* class using an association *hasPri-*

Figure 7. A conceptual HIT system model in UML via HL7 CDA



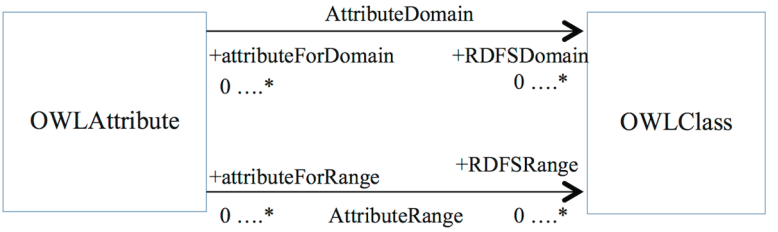
maryProvider. The *Patient* is also associated with the *Observation* class using the association *hasMedicalObservations*, the *Procedure* class is associated with the *Substance* classs with association *involvesSubstance*, *etc*. When this UML model is mapped to an OWL model (Baclawski, et al., 2002), the attributes *id*, *Weight*, *Height* are mapped to owl:DatatypeProperty, while the attributes *hasName*, *has Address, hasVitals, etc.,* and the associations *hasPrimaryProvider, hasMedicalObservations, performedProcedures, etc.,* are mapped to owl:ObjectProperty. As a result, there is no semantic differentiation shown between association (identified as *roles* in DL) and attribute, which can cause semantic ambiguity in representing a link between the concepts and results in a lack of a true "class" concept with attributes in OWL.

Our intent with the addition of *attributes* to OWL is to eliminate this semantic ambiguity in representing a relationship between classes and characteristics owned by a class. Therefore, in order to capture the essence of a class from a modeling perspective, we redefine the semantics of the entity *Class* in OWL and introduce a new first class element OWLAttribute (owl:Attribute) as shown in the Figure 8 to handle attributes and utilize the existing owl:ObjectProperty entity to capture the associations between the classes. The domain and range of the owl:Attribute entity is owl:Class, but is constrained and is placed at the same layer as OWLObjectProperty in the ODM property hierarchy as was shown in Figure 5. The semantics of the class can be defined as *Class* $\{At_0, At_1 \ldots At_n; Dt_0, Dt_1 \ldots Dt_n\}$, a set of attributes, where each At_i is the attribute and Dt_i is the datatype for all n define the class. For providing a syntactical and semantic definition consistent with the OWL 2 guide (OWL 2, 2004), the attribute can be defined as a *role* At_i $(A, B) \subseteq \Delta^I \times \Delta^I$ with additional constraint that there exists **no** $R (C, B) \subseteq \Delta^I \times \Delta^I$, where R is a binary predicate in the same domain of disclosure(Δ^I). Correspondingly, we can represent the same definition axiomatically as $At_i(A,B) \cap \notin R(C,B) \subseteq \Delta^I \times \Delta^I$, which also states that the concept A has an attribute concept B connected through the attribute At_i and the concept B can't be involved in a binary relationship R with any other concept such as C. For example, the concept *A* (Observation and Procedure) are connected to the concept *B* (Vitals and IVL) through attribute (hasVitals and hasEffectiveTime), and then there is no other concept *C* (Patient) that can form a relationship or role or association with *B* (Vitals or IVL).

The syntax of the attribute element is defined using the notation: <owl: Attribute rdf:ID="hasVitals"/>. Using this OWL attribute definition, and the structure as given in Figure 8, the Vitals class shown in the Figure 7 with attributes and can capture actual patient vitals (pulse, BP, respirations).

The patient vitals could be a crucial part of a study when researchers identify the required patient population for the research topics RT2.A, RT3.A, RT4.A, etc., as presented at the end of Section 2. Notice that each UML class has been translated to an OWL class, with the proposed owl:Attribute capturing the attribute *hasVitals* between the Observation and Vitals classes. The

Figure 8. The structure of an OWL Attribute model element



result as given in Figure 9 can be considered an *ontology schema* since it is defining the data types, structure, and associations.

To summarize, by introducing the concept of *attribute* for OWL, a *class* can be defined as a group of attributes, employ owl:ObjectProperty for representing associations, and define essentially a *schema* for the ontology as given in Figure 9. This is equivalent to a UML class diagram (Figure 7) which defines classes, attributes, and associations. As a result, rather than integrating ontologies at the instance level, it is now possible to integrate ontologies at a schema level, to allow associations across different ontology schemas to be mapped with integration rules. As a result, when one then attempts to integrate domain data from multiple ontologies, the mapping integration rules can be applied with the potential of alleviating or even eliminating semantic interoperability issues. This mapping across ontologies at the schema level is the subject of our ongoing research.

5.2. The Owl Domain Profile Extension

In this section, we define the OWL Domain Profile (ODP), our second extension to OWL, a feature for extending the primitive OWL meta-modeling elements for developing domain specific meta-modeling entities. Our intent with OWL Domain Profile is to capture abstract concepts that are initially laid out by the stakeholders in order to build the domain model. In UML, the developers employ UML Profile that provides a means to customize the UML meta-model to a particular domain. Generally, in software engineering, before developing domain models or schemas (M1 Level, Figure 6), stakeholders agree on a higher-level abstract theory based on which a domain model is realized. Once there is agreement on the abstract theory (i.e., abstract type concepts), the ontology designer traditionally focuses on the domain model concepts and its vocabulary at

Figure 9. The OWL Translation of UML diagram in Figure 7

```
<owl:Class rdf:ID="Patient">
<owl:Class rdf:ID="Provider">
<owl:Class rdf:ID="Disease">
<owl:Class rdf:ID="Symptom">
<owl:Class rdf:ID="Drug">
<owl:Class rdf:ID="Procedure">
<owl:Class rdf:ID="Name">
<owl:Class rdf:ID="EName">
```

```
<owl:Attribute rdf:ID="hasName">
<owl:Attribute rdf:ID="hasEName">
<owl:Attribute rdf:ID="hasName">
      <rdfs:domain rdf:resource="#Patient"/>
      <rdfs:range rdf:resource="#Name"/>
 </owl:Attribute>
<owl:Attribute rdf:ID="hasEName">
      <rdfs:domain rdf:resource="#Disease"/
>
      <rdfs:range rdf:resource="#EName"/>
 </owl:Attribute>
```

(a)

```
Class (a:Patient)
Class (a: Provider)
Class (a:Disease)
Class (a:Drug)
Class (a:Procedure)
Class (a:Name)
Class (a: EName)
```

```
AttributeAssertion (a:Patient a:hasName a:Name)
AttributeAssertion (a:Disease a:hasEName
a:EName)
AttributeAssertion (a:Symptom a:hasEName
a:EName)
.....
AttributeAssertion (a:Drug a:hasEName a:EName)
```

(b)

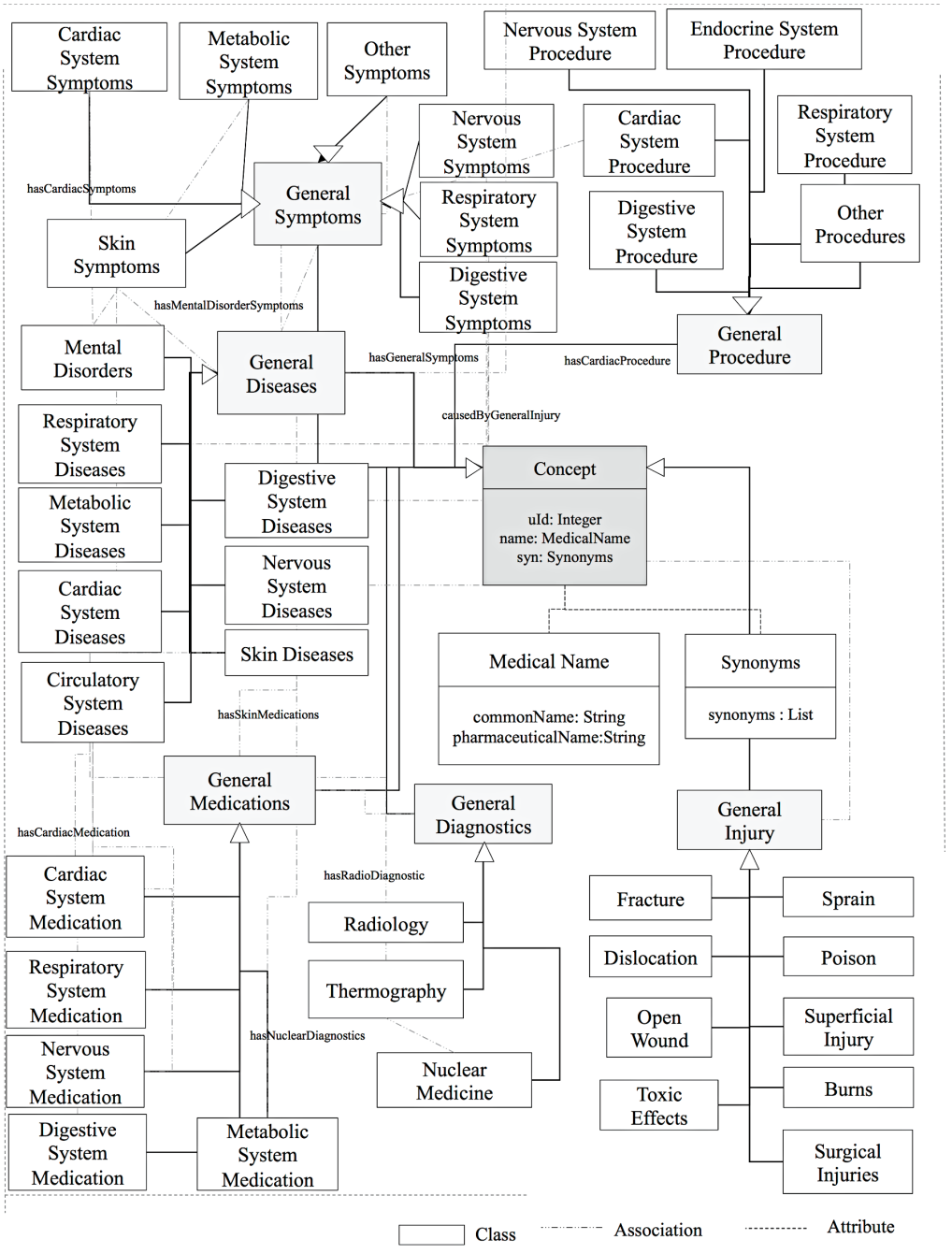
the instance level (e.g., actual terms and their structure in an ontology), neglecting to capture the agreed abstract theory and its type concepts. Our intent with ODP is to capture the abstract theory and its concepts that are initially laid out by the stakeholders in order to build the domain model.

For the conceptual understanding of abstract theory and type concepts, consider the sample ontology domain models developed using UML as shown in the Figures 10, 11, and 12. Briefly, Figure 10 presents the Diagnosis Ontology Model that captures the required knowledge for identifying a patient's conditions, its causes, and the approaches to mitigate these conditions. Figure 11 details the Test Ontology Model that captures the knowledge on various medical tests required for analyzing various patient conditions. Then, Figure 12 represents the Anatomy Ontology Model that captures knowledge on physical parts and their interactions of the human body. To fully explain, Figure 10 presents the Diagnosis Ontology Model that defines classes such as Metabolic System Diseases, Digestive System Diseases, Cardiac System Diseases, Skin Diseases, etc., which will each include attributes to represent what needs to be recorded for each class and defines and represents associations between the participating classes. These classes are used to realize the actual medical instances or vocabulary of the ontology such as digestive system disease instances, cardiac system disease instances, respiratory system disease instances, skin diseases instances, etc. Similarly, other classes with their respective attributes (such as hasCode, hasEffectiveTime, hasLimits, etc.) in the ontology model such as Respiratory System Procedure, Digestive System Medications, Fractures, Dislocations, etc., can be utilized to capture the respective medical vocabulary and are interconnected via associations such as hasCardiacSymptoms, hasGeneralSymptoms, hasMentalDisorderSymptoms, hasCardiacProcedure, causedByGeneralInjury, hasCardiacMedication, etc. Figure 11 details the Test Ontology Model that defines classes such as Physical Tests, Laboratory Test, Blood Test, etc., with respective attributes (hasCode, uId, MedicalName) to represent features of the class and associations (is-a, etc.) between the classes to capture the interactions between the defined classes. These classes are later realized to capture actual medical test instances or the vocabulary of the ontology. Similarly, the Anatomy Ontology model shown in the Figure 12 captures human anatomy using the class Human Parts with respective attributes to describe the human part and associations to capture various relationships between them. This model is later instantiated to capture instances of human parts such as Heart, Veins, Arteries, etc.

However, from the perspective of the meta-model (Level M2, Figure 6) and by using the exemplified ontology models in Figures 10-12, a ontology designer can abstract and define generic domain specific type concepts shown in Figure 13, that include: classes such as Metabolic System Diseases, Respiratory System Diseases, Digestive System Diseases, etc., which are all of type Disease; classes such as General Symptoms, Cardiac System Symptoms, Respiratory System Symptoms, etc., which are all of type Symptom; classes such as Cardiac System Medication, Respiratory System Medication, Nervous System Medication, etc., which are all of type Medication; classes such as Cardiac System Procedures, Respiratory System Procedures, etc., which are all of type Procedure; classes such as Cardiac System Test, Respiratory System Test, Blood Test, etc., which are all of type Test; classes such as Fracture, Sprain, Dislocation etc., which are all of type Injury; and, classes such as General Diagnostics, Radiology, Nuclear Medicine, etc., which are all of type Diagnostic.

Within these abstract type concepts (Figure 13), abstract attribute types can be defined, namely: hasPharmaceuticalName, hasCodeName, hasCommonName, etc., which are all of type hasName; and, hasUId, hasDeaNumber, etc., which all are of type hasId. Figure 13 illustrates the abstract associations types: hasGeneralSymptoms, hasMentalDisorderSymptoms, hasCardiacSymptoms, etc., which are all of type

Figure 10. Sample diagnosis ontology models developed in UML



hasSymptom; hasCardiacProcedure, hasRespiratoryProcedure, hasDigestiveProcedure, etc., which are all of type hasProcedure; hasBloodTest, hasPhysicalTest, performXRay, etc., which are all of type hasTest;(hasCardiacMedication, hasSkinMedication, hasDigestiveMedication,

Figure 11. A test ontology model in UML

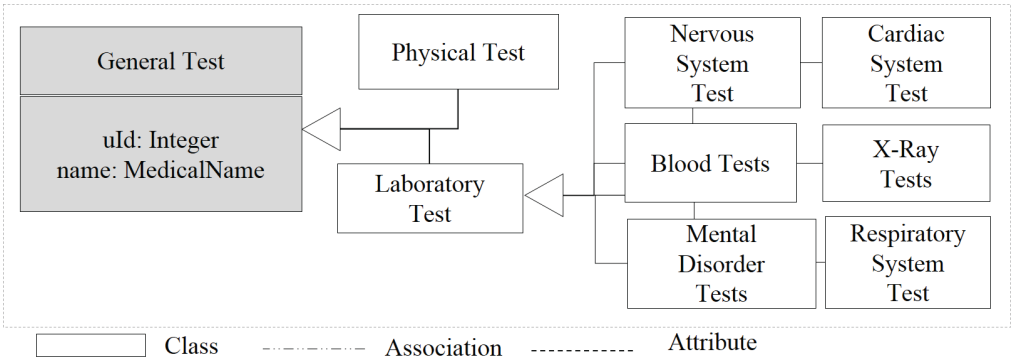


Figure 12. An anatomy ontology model in UML

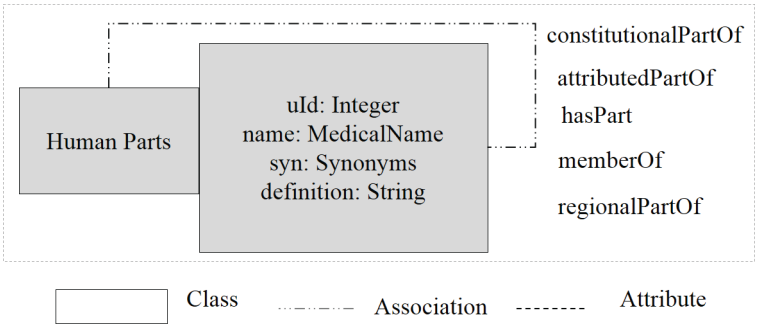
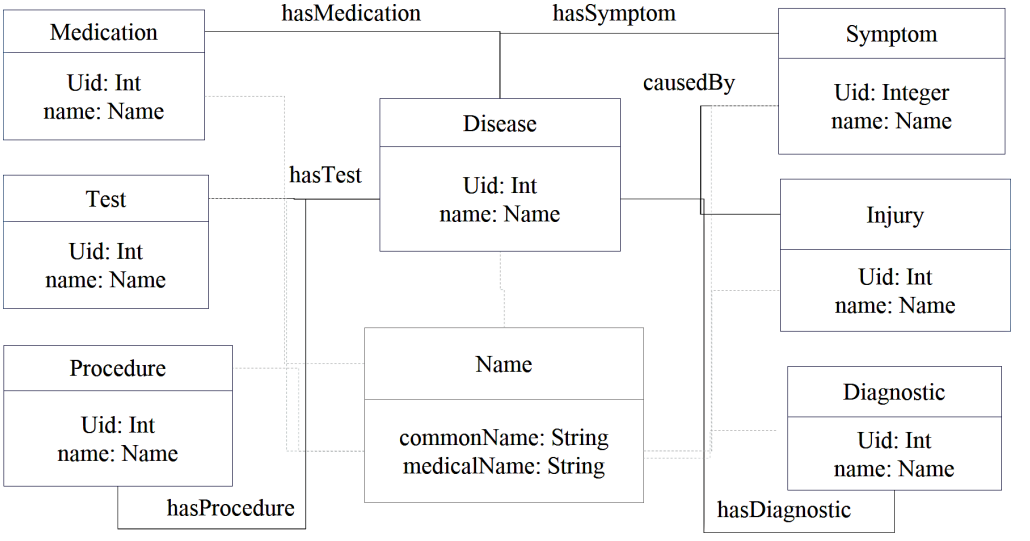


Figure 13. The abstract theory from ontology models shown in Figures 10-12



etc., which are all of type hasMedication; hasNuclearDiagnostic, hasRadioDiagnostic, etc., which are all of type hasDiagnostic; and, isCausedByFracture, isCausedByDislocation, sprainedBy, etc., which are all of type causedBy.

Essentially, the ontology models in Figures 10-12 are used as the basis to yield a more generalized knowledge base for the given domain. The result in Figure 13 is an abstract theory with type concepts that can be reused in multiple contexts for many HIT systems and EHRs for the development of multiple domain models.

For example, another domain ontology model can be developed involving: classes such as Ear Diseases, Swelling, Eye and Cornea Diseases, etc., which are of type Disease; associations such as hasEyeInfections, developsEarInfection, showsSwelling, etc., which are of type hasSymptom; and, attributes hasSynonym, hasBiologicalName, previouslyNamed, etc., which are of type hasName. This signifies that even though the ontology model (classes, attributes, and associations) expresses knowledge on different domains, both the domain models use the same abstract theory (Figure 13) for developing multiple ontology models as shown in the Figure 14a.

Thus, the sample abstract theory captured as a domain profile (Figure 13) at a higher level (meta-model level, Figure 6) can be used as a base theory to develop a global ontology model capturing knowledge about various domains such as diseases, symptoms, tests, medications, procedure, etc., or another biomedical ontology model which can meet the knowledge requirement of the researchers RT's. For instance, the knowledge required for RT1.A and RT1.B (Section 2) will involve types Disease and Medication with their respective association type (hasMedication) to obtain an output which can be of type Symptom or Injury, RT3.A requires types Disease and Test to obtain common test patterns between similar disease profiles, etc.

The proposed OWL Domain Profile (ODP) supports the OWL framework by allowing the ontology designer to capture the abstract type concepts as profile concepts akin to UML profiles at the meta-model level, and to impose (reuse) them onto the multiple ontology models (Level M1, Figure 6) as shown in Figure 14b. In order to impose the profile concepts onto the domain model entities and automate the load/parse/save process, we have designed and implemented a *DomainProfileParser* algorithm that authenticates and validates the imposing of the profile entities onto the ontology model concepts and structural associations. Figure 15 illustrates the comparison of ontology modeling using OWL and OWL supported with ODP to define the same (Figure 13) information using both OWL (Figure 15a) and our proposed extensions with ODP (Figure 15b). Figure 15a

Figure 14. An architectural perspective of OWL ODP and its domain models

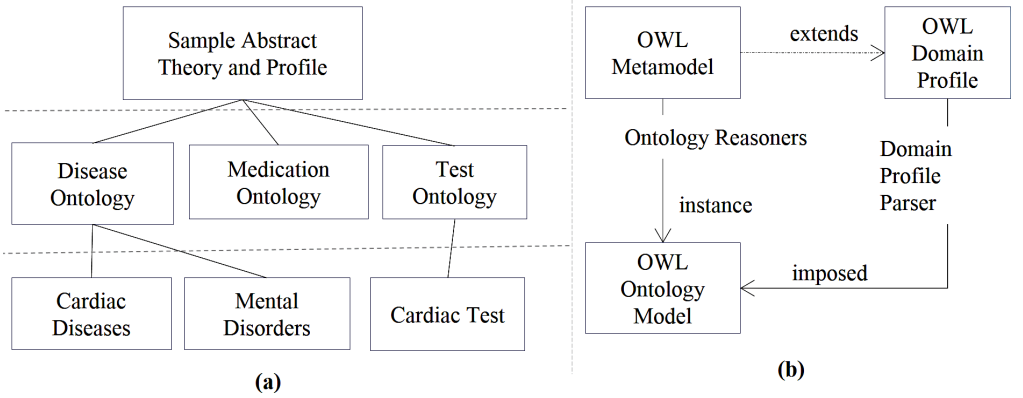
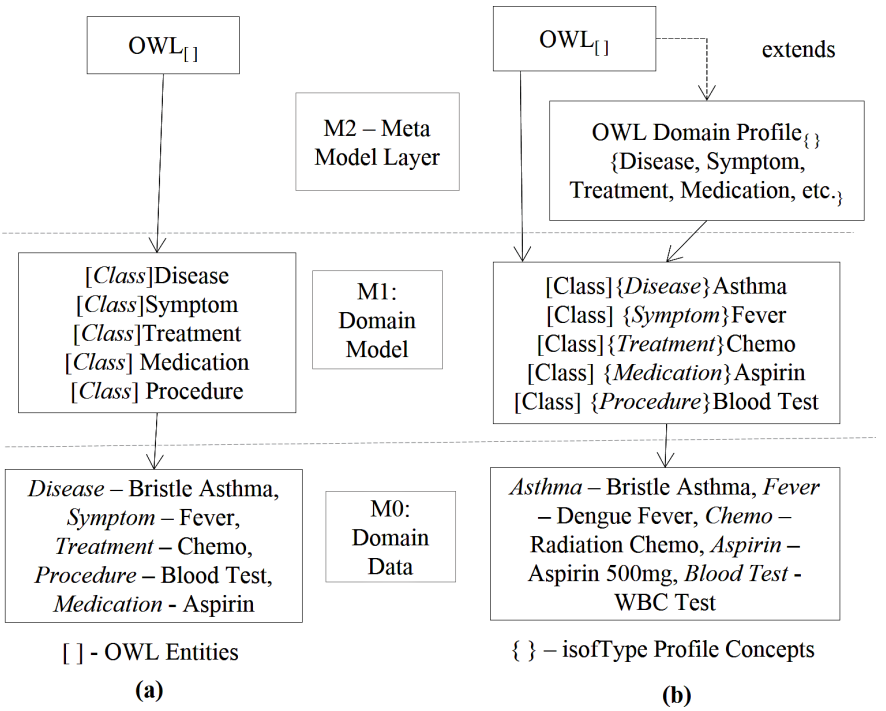


Figure 15. A comparison of modeling using OWL and OWL+ODP



demonstrates the development of a domain ontology using traditional OWL, where the concept Disease is defined as an owl:Class entity, which in turn is instantiated to define domain instances such as Brittle Asthma captured as owl:NamedIndividual. In contrast, in Figure 15b, OWL is extended with ODP, allowing a generic concept such as Disease to be abstracted as a profile concept (Sample Abstract theory, Figure 13) that is then imposed onto the concept Asthma (read as Asthma isofType Disease) defined as a class by instantiating owl:Class, which is turn is instantiated to define domain instances such as Brittle Asthma captured as owl:NamedIndividual. In general, when OWL is used alone, the owl:Class metamodel element is instantiated to create domain classes (applicable to a general HIT system's concepts) which are later instantiated for capturing domain data (for a specific vendor's HIT sytem's concepts). Using OWL extended with ODP, the defined ODP entities can be imposed onto the domain model that are then instantiated to capture instances; this is shown in the transition from the classes in Figure 15a to the domain profile entities in Figure 15b.

The ODP entities extend OWL primitive meta-modeling elements by creating a dependency relationship with the OWL meta-model and, hence, can only be utilized in the OWL meta-model framework as shown in the Figure 15b (M2 – meta-model level). In ODP at the meta-model level, there are four extensions: *Profile Class* (P_C) is utilized for capturing domain specific type concepts; *Profile Attribute* (P_A) is intended to capture the characteristics of a Profile Class, namely the comprising attributes; *Profile ObjectProperty* (P_{OP}) captures all of the interactions between profile classes, which may be inheritance or associations; and, *ProfileDatatypeProperty* (P_{DP}) defines properties to capture datatype values (such as integer, URI, String etc.) for a profile class. From a practical perspective, these four extensions of

ODP extend OWL first-class entities: owl:Class, owl:Attribute, owl:ObjectProperty, and owl:DatatypeProperty to define ODP entities odp:ProfileClass(P_C), odp:ProfileAtrribute(P_A), odp:ProfileObjectProperty(P_{OP}) and odp:Pro fileDatatypeProperty(P_{DP}), respectively. The primary objective of ODP is to encapsulate the abstract theory which can be reused across multiple settings (instantiated differently for each vendor's HIT System). A secondary objective is to leverage P_C, P_A, P_{OP}, and P_{DP} for defining meta-schema at M2 Level which can be referred to when developing ontology models at M1 Level in Figure 6. The remainder of this section reviews P_C, P_A, P_{OP}, and P_{DP}:

Profile Class (P_C): A *Profile Class* is an ODP meta-modeling entity that is represented using the tag odp:ProfileClass for developing domain specific class meta-model (M2 Level, Figure 6) entities such as *Disease, Virus,* etc., by extending OWL's owl:Class primitive element. A *Profile Class* extends the core element owl:Class but does not change the semantics of the element. As given in Figure 16, *Disease*, *Procedure*, *Treatment*, *Symptom* and *Medication* are of type ProfileClass, which are imposed onto the domain model concepts with our extended ODP syntax;

Profile Attribute (P_A): A *Profile Attribute* is an ODP meta-modeling entity utilized for defining domain specific attributes such as *hasSymptom*, *hasScientificName*, *hasICDCode*, etc., by extending owl:Attribute element. A Profile Attribute is represented by the tag element odp:ProfileAttribute which extends owl:Attribute element, but does not change the semantics of the core element. From Figure 17, *name* is a odp:ProfileAttribute with domain *Disease* and range *EName*, with our extended ODP syntax in Figure 17;

Profile ObjectProperty (P_{OP}): The *ProfileObjectProperty* is represented by the tag element odp:ProfileObjectProperty is a member of the ODP for encapsulating abstract domain specific roles (Figure 13) such as *hasProcedure, hasMedication, hasSymptom, hasTreatment,* etc. ProfileObjectProperty follows the semantics of owl:ObjectProperty for capturing associations. A ProfileObjectProperty can only capture interactions between entities of type odp:ProfileClass, with our extended ODP syntax in Figure 18;

Profile DatatypeProperty (P_{DP}): The *ProfileDatatypeProperty* captures domain specific roles whose range is a datatype such as integer, double, URI, time, etc. This element extends OWL's owl:DatatypeProperty and is represented using with element odp:ProfileDatatypeProperty. ProfileDatatypeProperty follows the semantics of owl:DatatypeProperty and takes domain values only of type odp:ProfileClass. The datatype *uid* from Figure 13 is of type ProfileDatatypeProperty with our extended ODP syntax in Figure 19;

In summary, the ODP provides a means to transition to a higher level of conceptualization that promotes ontology design. As a result, as illustrated in Figure 13, 14 and Figure 15b, stakeholders can agree on ontology meta-schema details at a higher conceptual level and create

Figure 16. Sample code illustrating the profile class

```
<odp:ProfileClass rdf:ID="Disease"/>
    <odp:Disease odp:isofType = "Asthma"/>
    <Astham rdf:id="Brittle Asthma"/>
<odp:ProfileClass rdf:ID="Procedure"/>
    <odp:Procedure odp:isofType = "Chemo"/>
    <Chemo rdf:id="Radiation Chemo"/>
<odp:ProfileClass rdf:ID="Medication"/>
    <icd:Medication odp:isofType = "Aspirin"/>
    <Aspirin rdf:id="Aspirin 500mg"/>
<odp:ProfileClass rdf:ID="Symptom"/>
    <icd:Symptom odp:isofType = "Fever"/>
    <Fever rdf:id="Dengue Fever"/>
```

Figure 17. Sample code illustrating the profile attribute

```
<odp:ProfileAttribute odp:ID="name"/>
   <odp:ProfileDomain rdf:resource="Diseases"/>
      <odp:ProfileRange rdf:resource="EName"/>
</ odp:ProfileAttribute>
<odp:EName odp:isofType="medicalName">
```

Figure 18. Sample code illustrating the ProfileObjectProperty

```
<odp:ProfileObjectProperty  odp:ID="hasSymptom"/>
      <odp:ProfileDomain > Diseases </odp:ProfileDomain >
      <odp:ProfileRange> Symptom </odp:ProfileRange>
</ owl:ProfileObjectProperty >
<odp:hasSymptom  odp:isofType = "hasCardiacSymptom"/>

<odp:ProfileObjectProperty  odp:ID="hasTreatment"/>
      <odp:ProfileDomain > Diseases </odp:ProfileDomain >
    <odp:ProfileRange> Treatment</odp:ProfileDomain >
</ owl:profileObjectProperty >
< odp:hasTreatment  odp:isofType = "hasChemoTreatment"/>
```

Figure 19. Sample code illustrating the profile data type property

```
<odp:ProfileDatatypeProperty rdf:ID="hasUid"/>
      <odp:ProfileDomain> Disease </odp:ProfileDomain>
      <odp:ProfileRange>&xsd;Integer</odp:ProfileRange>
</ odp:ProfileDatatypeProperty >
<odp:hasUid odp:isofType ="hasId"/>
```

type concepts that are reusable. Once the abstract type concepts are captured as profile concepts, they can be employed by different HIT system, where each HIT system would build their own ontology model based on their specific data, but all of the systems would share the same abstract theory (same generalized meta-schema) at the meta-model level, thus reusing the core conceptual abstract theory and type concepts.

5.3. The OWL/OMV Schema Associations Extension

The primary goal for developing *Ontology Schema Associations* or *Ontology Schema Relationships* for OWL ontology models is to capture appropriate associations between and across ontology models. Current usage of OWL when developing ontologies often embeds these associations within a single ontology. As a result, an ontology that has diseases referencing symptoms referencing treatments can be searched in that structured order, but, if one wants to find all diseases that have the same treatments, such a search is very difficult to write. With ontology schema associations, there would be separate ontologies for disease, symptoms, and treatments, that can be linked in different ways to allow a wider variety of queries using a higher-level structure that doesn't bury linkages deep with an ontology tree.

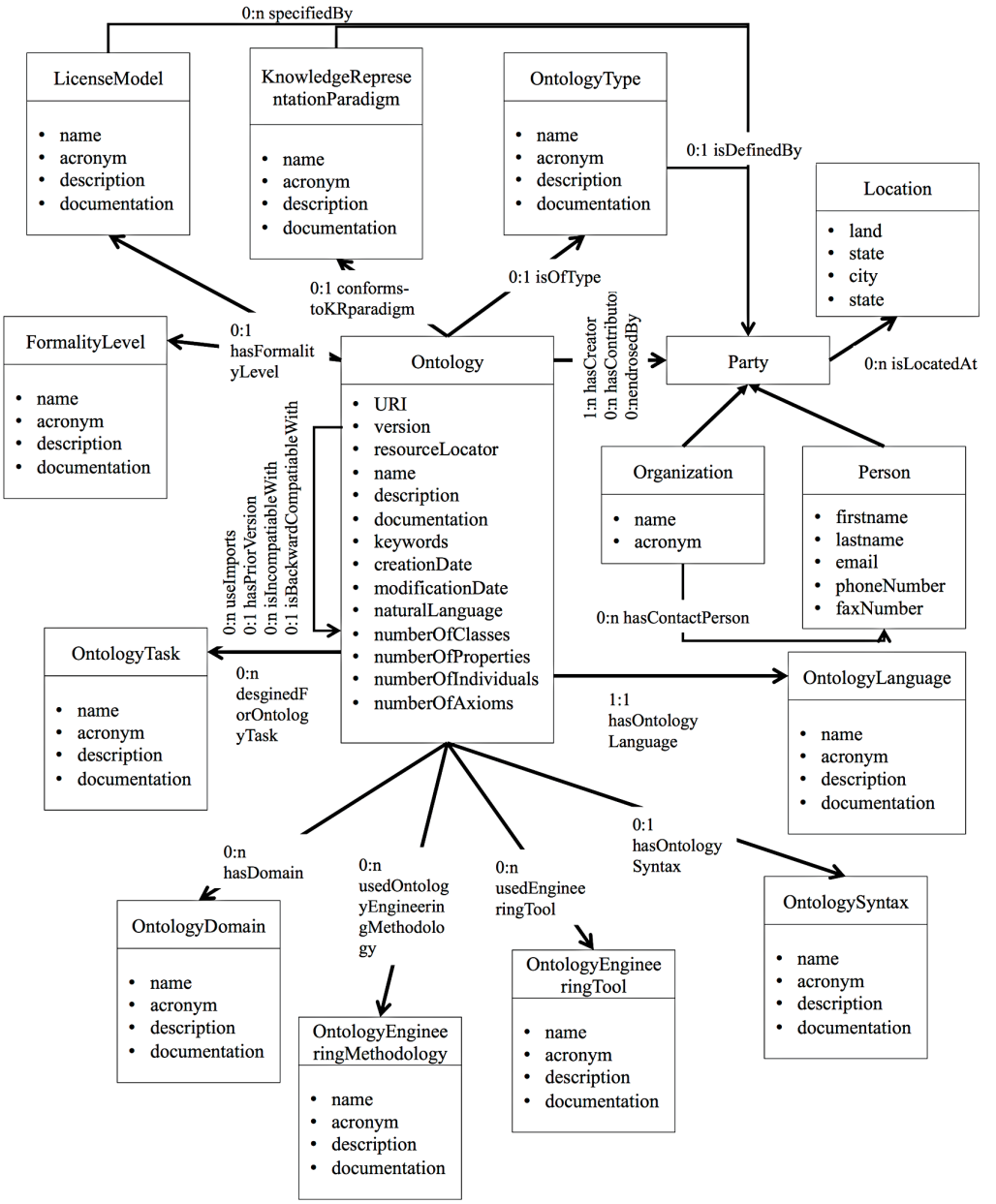
The first two extensions for Attribute (Section 5.2) and OWL Domain Profile (Section 5.2) are focused on the OWL features and capabilities

applied at the metamodel and domain model level to dictate the way that domain models are developed. The intent of *Ontology Schema Associations* is to leverage realize the Ontology Meta Vocabulary (OMV) (Hartmann, Palma, & Sure, 2005) model using OWL framework, which is then instantiated for each ontology model to capture the respective metadata and later interrelated with another ontology model metadata concepts to define schema associations. The OMV is a domain model for providing metadata for the ontology and was developed as part of NeOn project group. OMV provides a way to capture the metadata for the ontology as related to *domain, organization, language, place, version, tools,* etc. To demonstrate, Figure 20 provides an overview of core OMV concepts. In Figure 20, in a typical ontology engineering process has a *Person(s)* or *Organization(s)* responsible for developing an ontology represented using *Ontology* entity can be grouped to form a generic entity *Party* by a subclass-of relation. The geographical information for the party is described by the *Location* entity. Further, the engineering process employed for building the ontology is captured by *OntologyEngineeringMethodology* and development tools, namely, *OntologyEngineeringTool*. The entity *OntologyTask* describes the primary task of the ontology and *LicenceModel* describes the usage boundaries of the ontology schema. The entities *OntologySyntax*, *OntologyLanguage* and *KnowledgeRepresentationParadigm* capture the implementation details of the ontology schema. A further classification of the ontology schema relies on their level of formality captured using *FormalityLevel* and types of knowledge representation primitives supported using *KnowledgeRepresentationParadigam*. The domain the ontology describes is represented by the entity *OntologyDomain,* while *OnotologyType* describes the nature of the content of the ontology.

Our intent for leveraging OMV is to allow *ontology associations* at the schema level. For instance, consider a medical domain that needs an ontology that contains information on *diseases, medications, treatments, symptom, assessment*, and *test.* In a traditional ontology approach, an ontology designer would put together these concepts into single source ontology (Os), and say organize the concepts first by *disease*, then *symptoms* of each disease, along with *treatments, medication, assessment* and *test* for each disease. If one only wants to reuse the *disease, tests* and *assessment* concept models, it may involve a complex ontology transformation or importing the complete source ontology, which also imports unwanted concepts. This becomes a huge issue when dealing with large ontologies (thousands of concepts) which have a direct impact on performance as the application parser and reasoners also needs to import, read and parse unwanted concepts. In order to address this issue, our approach as shown in Figure 21, is to separate *Disease, Tests*, and *Assessment* concepts into independent ontology schemas and then interconnect the schemas using relationships. As shown in Figure 21, the Diagnosis Ontology Schema (O_1) links *Disease* to *Symptoms*, *Treatment*, *Procedure,* and *Medication*; a Test Ontology Schema (O_2) describes various tests such as *Cardiac Test*, *Blood Test*, *Image Test,* and *Respiratory Test* etc.; and, a Triage[1] Ontology Schema (O_3) provides initially *assessment* about *vitals* and previous *injuries* and also may provide *medication* information. Each of these ontology schemas can be described (metadata about the ontology) by associating various OMV concepts. As shown in Figure 21, the ontology schema O_1 has OMV concept ontologyDomain with value "Condition", O_2 has ontologyDomain with value "Triage" and O_3 has the concept ontologyDomain with value "Test". These OMV concepts across these individual ontology schemas (O_1, O_2, and O_3) are associated to form *Ontology Schema Associations*.

By separating the domain model into multiple schemas, pieces of the domain model can be reused in other applications, much as a subset of a UML class diagram can be utilized in different solutions of a similar domain. This contrast with a single ontology that may either be unable to represent the knowledge requirement

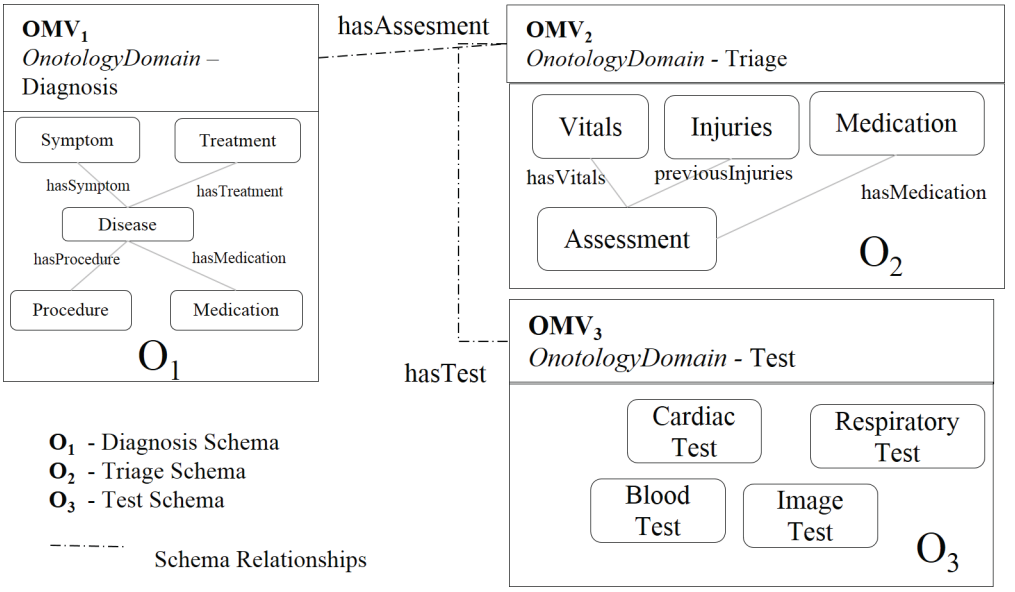
Figure 20. An overview of OMV core concepts



of the system and also be incapable of being seperated easily to be reused in other contexts, as we described above. Trying to capture entire knowledge within a single ontology will sacrifice modularity and the potential for reuse.

Figure 21 shows hasAssessment and hasTest which are Ontology Associations which are binary predicates between OMV concepts across multiple ontology schemas as indicated by the dotted lines. Ontology Associations are

Figure 21. Illustrating ontology relationships between ontology schemas



similar to package imports and class diagrams in UML and schema reference in XML. OWL also has the ability to reference other ontologies, but with OMV, ontology designers can define customized ontology relationships between OMV concepts across multiple ontology models. Besides providing modularity and knowledge reuse, schema associations assist the user to identify required knowledge sources through ontological relationships for RT's. For instance, while a user is researching on side effects of hyperglycemic agents on CHF and diabetes (RT2.B, Section 2) using the Diagnosis Ontology, the researcher might be interested in also knowing the various tests which can be performed for conducting statistical analysis. The schema association hasTest between the Diagnosis and Tests ontologies assists the researcher in identifying the appropriate ontology for obtaining medical test knowledge. Similarly, for finding the physical impact of diabetes while using metformin (RT3.A, Section 2) can be explored using knowledge captured in anatomy ontology related to diagnosis ontology using effects schema associations.

6. AN ONTOLOGY DESIGN AND DEVELOMPENT LIFE CYCLE

In this section, we briefly review our work on a software development life cycle for ontologies that provides the design methodology as characterized in Figure 1 in support of a design process for the three OWL extensions for Attribute, Domain Profile, and Schema Associations as discussed in Sections 5.1, 5.2 and 5.3, respectively. Historically, there are numerous life cycles including the waterfall model (Winston, 1970), the iterative model (Basili & Turner, 1975), the spiral model (Boehm, 1986), agile development (Craig, 2003), and others, that share phases such as requirements, specification, design, analysis, implementation, testing, deployment and maintenance. For ontologies, there have been related efforts that are of note. Methonotology (Fernández-Lopez, Gomez-Perez, & Juristo, 1997) employs phases: specification for knowledge acquisition to develop an ontology vocabulary; and, conceptualization to structure the domain vocabulary and develop a conceptual model for the ontology, integration,

implementation and evaluation. The approach leverages an evolutionary model to expand the ontology over time as new requirements emerge and was based on earlier work that proposed a method based on steps for requirements and specification, construction of data dictionary, concept classification, and other instance-based ontology considerations. Another effort (Uschold & King, 1996) proposed an enterprise ontology with requirements, conceptualization, implementation, and documentation phases that was later expanded (Uschold, 1998) in order to create a unified methodology.

In this section, we review our work on a Hybrid Ontology Design and Development Life Cycle (HOD²LC) model (Saripalle & Demurjian, 2012) that leverages various software engineering process and methodologies which can be applied to ontologies. The intent is to provide a concrete context for utilizing the OWL extensions coupled with the alignment with the meta-model to yield a design process. The HOD²LC model has a number of phases that represent different aspects of the ontology design and development process, and integrate with approach (Figure 1) and proposed extensions (Sections 5.1 to 5.3):

Phase 1: The *Problem Analysis* phase identifies and analyzes the problem faced in information system leading to the development of new ontology and/or extending an existing ontology. Generally the problems faced are related to instance data of the domain or a state of the ontology, from which an abstract domain problem has to be formulated. For example, list all of the symptoms of Radiation Chemotherapy on Breast Cancer; when rewritten into a domain problem, the query is focusing on domains of *symptoms*, *treatment* and *diseases*. The problem analysis is to identify the abstract meta-concepts (M2 Level, Figure 6) and the domain model concepts (M1 Level, Figure 6) from instance data (M0 Level, Figure 6);

Phase 2: The *Integration Phase* allows designers to search for existing ontologies meeting the problem criteria. For instance, reusing *RxNorm* (Liu, Ma, Moore, Ganesan, & Nelson, 2005) which provides normalized names for clinical drugs and UMLS provides semantic types and network (Bodenreider, 2004) to in support various medical concepts;

Phase 3: In the *Knowledge Acquisition phase*, designers interact with domain experts (providers, researchers, etc.) searching multiple resources (medical records, data, ontologies, etc.) to identify the concepts and domain vocabulary required to develop the complete ontology. This phase can be performed in parallel with *Specification*, *Design* and/or *Analysis* phases;

Phase 4: In the *Specification phase* the designer firmly defines the domain's scope and functional details of the ontology and its concepts. For instance, the *disease* domain from Phase 1 can be refined by specifying types of *diseases*, associating *diseases* with *symptom*(s) and name as its attribute (see Figure 7), resulting in a conceptual schema for disease and symptom and then associating these schemas;

Phase 5: In the *Design Phase*, the concepts in the domain are identified, including: classes, attributes, and associations that can be classified into meta-concepts (Figure 13) and model concepts (Figures 10-12). For instance, the type concept *Disease* and *Symptom* (Figure 13) can be meta-concepts with *hasSymptom* a meta-association to capture their interactions. The meta-concepts can further be refined to identify domain model concepts (Figure 15);

Phase 6: In the *Analysis phase*, designers and the end users (and domain experts) revisit the specification phase to validate (all required system requirements are met) the design models developed in the Design Phase. The cycling between *Specification, Design, and Analysis phases* promotes an incremental learning process;

Phase 7: The *Implementation phase* provides the transition from conceptual model (UML Class Diagram or ERD diagram) to concrete

implementation (using OWL DL, OWL Lite, Frames, etc.). This phase requires decisions to be made regarding the particular ontology language and framework for the implementation, which can be based on issues related to usability, performance, interoperability, and availability;

Phase 8: The *Testing phase* carries out a technical judgment of the ontologies, their software environment, and documentation with respect to a frame of reference (in our case, the requirements specification document) during each phase and between phases of their life cycle, using techniques such as Ontolingua (Gomez-Porez, Juristo, & Pazos, 1995) or a framework for evaluating knowledge sharing technology (software, ontologies and documentation) (Gomez-Porez, 1996);

Phase 9: The *Maintenance and Documentation* phase is where the developed ontology has to be monitored for smooth and efficient performance of the system (maintenance) backed by a detailed narrative report of the ontology concepts, its axioms, and usage (documentation). This phase can start with knowledge acquisition and run in parallel with subsequent phases.

To form the complete life cycle model, Figure 22 illustrates a Hybrid Ontology Design and Development Life Cycle (HOD^{2}LC) involving the aforementioned phases. The HOD^{2}LC is an agile methodology through Phase 2 to Phase 7. The *iterative* and *incremental* approach assists developers to take advantage of what was learned during previous phases. There is also a need of loop between *Analysis* and *Specification phases*, as the developers have to validate the ontology model to check if the specifications have met. This loop is represented using a dotted line differentiating it from the life cycle's solid line. This loop helps the end users to verify or edit specifications so as to make necessary changes to the ontology model without go through the whole cycle. The *Knowledge Acquisition phase* can be executed in parallel with other phase until the *Implementation phase*, which is responsible for developing the vocabulary of the domain from the information gathered from the *Knowledge Acquisition phase*. The *Documentation phase* of the ontology can also be executed in parallel starting from *Analysis* phase. To support the iterative process employed in the *Design phase*, we utilize Feature Driven Development methodology (*FDD*) (Palmer & Felsing, 2002), a model driven agile software development process. Abstracting out the steps from FDD and applying it to our approach (Figure 23), we have the following steps. In *Step 1*, a higher-level walkthrough of the domains involved in the domain problem should be performed to identify type concepts (Figure 13). For example, type concepts such as *Disease*, *Symptom*, *Medication*, *Treatment*, etc.; type concept attributes such as *uid*, *name*, etc.; and type concept associations such as *hasSymptom*, *hasTreatment, hasMedication, hasParent*, and *isa*. This step is equivalent to identifying domain profile concepts at the meta-model level (M2 Level) as shown in Figure 15. In *Step 2*, once there is agreement on the abstract theory and its type concepts, they are decomposed into smaller domain concepts by multiple groups. For example, classes such as *Respiratory Diseases,* and *Cardiac Diseases* can be defined which are of type *Disease*; *Cardiac Symptoms* and *Mental Disorder Symptoms*, etc., of type *Symptom*. The respective attributes and associations are also identified. Finally, in *Step 3*, as the respective models have been built, the modular models can be interconnected using ontology schema associations to form a network of ontology models (Figure 21). The iterative nature of the cycle will help developers learn from the previous phase and the incremental nature shows the sign of progress and partial output to the end users. The cycle is stopped once an agreement has been reached on structural and semantic aspects of the ontology. The work presented in this section is our initial effort to quantify an ontology design and development process; our work is ongoing in this area to fine tune the process and apply to more complex, realistic examples.

Figure 22. HOD²LC, the hybrid ontology design and development life cycle

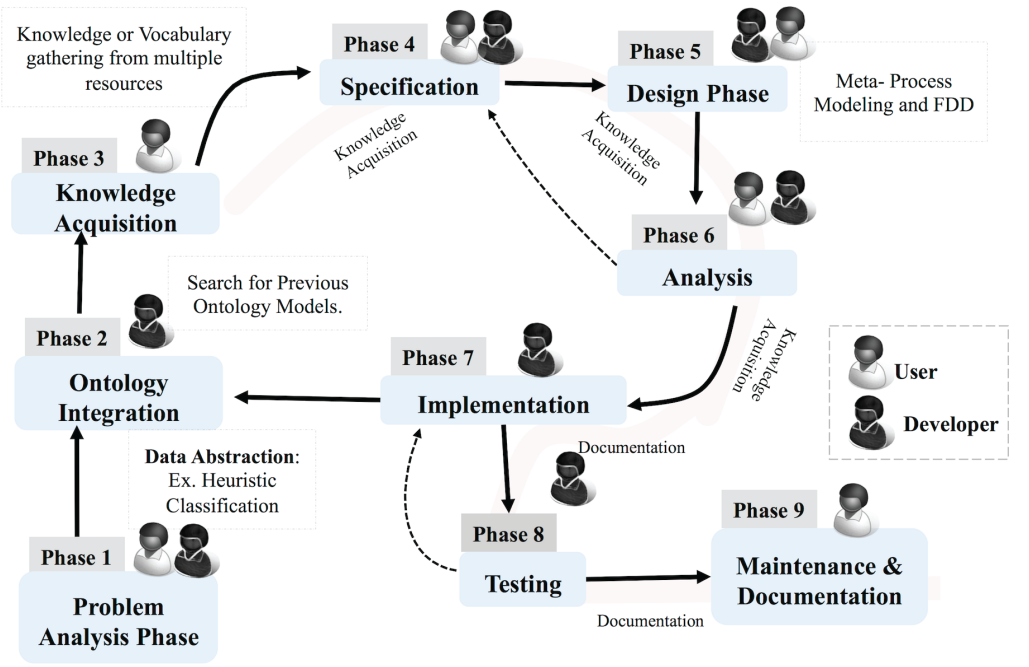
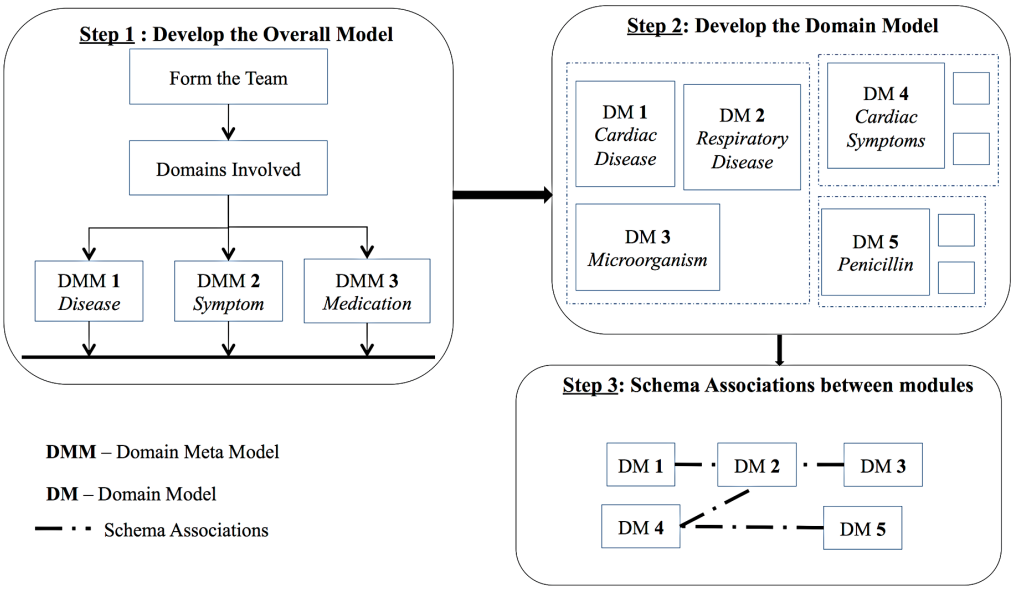


Figure 23. A feature driven development (FDD) of the ontology design phase



7. RELATED WORK

There are many related works that have influenced our objective of driving ontology development towards a model drive approach (MDA) that leverages software engineering. Gruniger and Fox (1995) has proposed a methodology to develop or enhance the ontology by formulating application scenarios which often are problems encountered in an enterprise application, giving rise to informal competency questions. The underlying existing ontology is enhanced to solve the problem by making sure that the queries against the ontology satisfy the competency questions. Bendaoud, Rouane-Hacene, Toussaint, Delecroix, and Napoli (2007) have proposed a methodology for developing ontologies using formal concept analysis. Similarly in Aussenac-Gilles, Biebow, and Szulman (2000), Bendaoud, Napoli, and Toussaint (2005), and Cimiano, Hotho, and Staab (2005), the research methodologies primarily focus on developing/enriching underlying application ontology concepts or roles between the concepts, but not on enhancing the core modeling capabilities of the ontology language. All of these efforts have the underlying intent of our work to improve the ontology design, development, and deployment process, but our work is more focused on extending OWL and ODM and leveraging OMV for a more software engineering-based approach.

The Network Ontology Model (NeOn) has a primary purpose to define a metamodel for integrating heterogeneous ontologies together by defining an OWL metamodel by extending MOF. The project also encompassed metamodels for SWRL, Rules, Mapping, and OMV. We have clearly leveraged their work as part of Section 5. Similarly, the work of Baclawski, Kokar, Kogut, Hart, Smith, Letkowski, and Emery (2002) has studied OWL and extended UML to provide visual models for developing ontologies. However, they found that the extensions were too complicated to understand and difficult to implement, and hence they proposed a Unified Ontology Language (UOL). The author's intentions were to provide a modeling environment for OWL similar to UML and in this process they *mapped* OWL language constructs onto UML concepts; this is comparable to our work, but, we differ since we want to try to incorporate additional capabilities into OWL and ODM rather than define an entire new model. The research presented in ODM (2009) discusses the development of an Ontology Definition Model (ODM) built on the top of MOF for enabling MDA for ontology engineering. This effort primarily focus on providing an MDA approach by extending MOF 2.0, but does not provide in-depth analysis about the modeling capabilities of OWL when compared to the UML software modeling techniques; our approach as presented in Section 3 seeks to leverage MDA and MOF to allow a similar process for ontology design, development, and deployment. The work of Kuhn (2010) supports our argument to advocate that the primary goal of an ontology engineer is to encode as much knowledge as possible and to exploit automated reasoning or discovering implicit knowledge, while acknowledging the lack of the essence of software modeling; like us, the work strongly recommends separating modeling and encoding concerns as modeling semantics as a design task while encoding is implementation oriented. This is also supported in Guranio (2001) that proposed the definition of an ontology as a *commitment* by a *language* (***L***) for capturing the intended meaning of the conceptualization. The intent of the *ontology* is to define a set of axioms in **L** such that the set captures the appropriate best possible model of conceptualization. The definition capture two aspects model and axioms, clearly emphasizing a design task and an implementation task, respectively.

Recently, researchers have been focusing on extending the OWL grammar with metamodeling capabilities Motik (2005). Recall that the OWL family encompasses three models (OWL Lite, OWL DL and OWL Full) arranged in ascending order of their expressiveness. OWL Full, the most expressive language supports metamodeling since a *class* can be viewed as group of instances or individual by itself, i.e., a class can refer to another class as if the latter is an *instance* of the former, which is basic

essence of metamodeling, and similar to our intent to associate ontology schemas. However, the underlying semantics of OWL Full are undecidable making it difficult for automated reasoning engines. The work of Motik (2005) demonstrated that OWL Full semantics are undecidable for metamodeling and bestowed metamodeling capabilities in OWL DL in two ways: *contextual semantics* and *Hilog semantics*. Similarly, the work in Glimm, Rudolph, & Völker (2010) explained the need to add metamodeling capabilities to OWL framework. However, both efforts focus on providing metamodeling capabilities to OWL, but not on extending existing OWL metamodel elements to define profiles and capture domain specific entities (Section 5.2).

The latest version of OWL 2.0 has introduced OWL Profiles (OWL 2 Profile, 2012), which is essentially a reduced version of complete OWL 2 semantics that trades expressive power for the efficiency of reasoning. OWL 2 has three profile formats: OWL2 EL for applications employing ontologies that contain very large numbers of properties and/or classes; OWL2 QL for applications that use very large volumes of instance data, and where query answering is the most important reasoning task; and, OWL2 RL for applications that require scalable reasoning without sacrificing too much expressive power. These OWL profile languages have restricted semantics from OWL, while the OWL Domain Profile proposed which is similar to UML profile, is a structural aspect acting in parallel with OWL (similar to UML Profile which acts according to UML) as shown in the Figures 17 and 18. The semantics and the decidability of the OWL language are unchanged. In the Software Design process, various methodologies and methods have been proposed by Fernandez-Lopez, Grüninger, and Fox (1995), Gomez-Porez (1996), Fernández-Lopez, Gomez-Perez, and Juristo (1997), Uschold and King (1996), and Uschold, (1998), but they are focused around the conceptualization phase and are attempting to maximize the domain vocabulary of the ontology, while they do not address the conceptual design phase and complete life cycle model, which are prominent in the software process and in our approach. Also, the design phases in these techniques are primarily centered on developing domain models using metamodels such as UML, RDF, and OWL, etc., but not a well-proven layered architectural approach such as UML Profile, DOGMA, and ODP. The *Analysis* phase (see Section 6) which is a crucial phase in the software engineering primarily responsible for analyzing the conceptual model and validating the model with specification, is not a part of the development cycle of many of these efforts, or has at best given just a limited consideration. UPON (Nicola, Missikoff, & Navigli, 2005) is an instance of Unified Process consisting of four phases (inception, elaboration, construction, and transition) and each phase is furthered iterated over five workflows (requirements, analysis, design, implementation, and test). The HOD^{2}LC is *agile methodology* over various enhanced phases (without cycles and workflows) and also leveraging proven software design techniques: meta-schema (ODP, Section 5.3) and the FDD approach (Figure 23) in our Design Phase and inner feedback loops in the Analysis Phase, and Testing Phase.

8. CONCLUSION AND ONGOING RESEARCH

This paper has addressed the issue of the current approach to ontology design, development, and deployment that is primarily focused on encoding the concepts and relationships directly at the instance level rather than fostering a model-driven approach like UML, ERD, and XML in which structural and relational artifacts of the data model are captured and the instance data follows the model rules. Towards this goal, in Section 3, we have reviewed our previous work (Saripalle, Demurjian, & Behre, 2011) and re-defined critical modeling characteristics and evaluated them against UML, ERD, and XML. Using this as a basis, in Section 4, we

explored domain modeling for UML using a layered OMG's Meta Object Facility (MOF) and then applying this approach in concept to XML, RDF, and OWL to demonstrate a more software engineering like process approach that aligns RDF and OWL to the layers of MOF. The main contribution of the work presented herein is in Section 4 with two extensions attribute (Section 5.1) and domain profile (Sections 5.2). With Ontology Metadata Vocabulary (OMV) framework and the ability to model ontology relations in Section 5.3. Both Sections 4 and 5 utilized an example in clinical informatics as presented in Section 2, to demonstrate the concepts. To place the work in this paper into a large concept, a software life cycle model is proposed via the Hybrid Ontology Design and Development Life Cycle (HOD²LC) in Section 6 which provides a process for ontology design, development, and deployment using OWL. We believe that the work presented herein is an important step towards a model-driven ontology design and development process that leverages software engineering principles and practices. The interested reader is referred to http://www.engr.uconn.edu/~steve/KanthSaripalle.html for supplemental material on implementation details of the work in this paper that integrate our OWL extensions into the Protégé tool (Protege, 2012).

Our ongoing research seeks to work towards a framework for ontology integration that operates at an ontology schema level and our contribution in Sections 5.1, 5.2, and 5.3, can be utilized to view ontologies at a schema level. If we are in a specific domain, say medicine, where we need to integrate ontologies from multiple HIT systems, the ability to define ontology schemas for each that break apart a traditional single ontology, and merging at the schema rather than the instance level, which has the potential to fully automate ontology integration. Additionally, with the ability to extend metamodel entities, we can capture generic domain concepts of medicine in a profile and impose them across multiple HIT settings. We are also considering and expanding UML via its metamodel with extensions to support an ontology design and development process integrated into UML. Our prior work on security access-control model (Pavlich-Marsical, Demurjian, & Michel, 2010) and collaborative security (Berhe, Demurjian, Gokhale, Pavlich-Mariscal, & Saripalle, 2011) extensions to existing UML and proposed new UML diagrams in support of role-based, discretionary, mandatory, and collaborative access control that were integrated into a UML setting, allowing for a comprehensive design process that included security. We believe such an approach would also benefit our work by including ontology design and development directly into the software process as another facet of overall information systems design.

REFERENCES

Allemang, D., & Hendler, J. (2011). Semantic web for the working ontologist, second edition: Effective modeling in RDFS and OWL. (2nd, Ed.) Morgan Kaufmann.

AllScripts. (2012). *AllScripts*. Retrieved from http://www.allscripts.com/

Aussenac-Gilles, N., Biebow, B., & Szulman, S. (2000). Revisiting ontology design: A method based on corpus analysis. In *Proceeding of International Conference in Knowledge Engineering and Knowledge Managment* (pp. 172-188).

Baader, F., McGuinness, D., Nardi, D., & Patel-Schneider, P. (2005). *The description logic handbook: Theory, implementation and applications*. Cambridge University Press.

Baclawski, K., Kokar, M., Kogut, A. P., Hart, L., Smith, E. J., Letkowski, J., & Emery, P. (2002). Extending the unified modeling language for ontology development. *Journal of Software and System Modeling*, *1*, 142–156. doi:10.1007/s10270-002-0008-4

Basili, V., & Turner, J. (1975). Iterative enhancement: A practical technique for software development. *IEEE Transcation on Software Engineering*, 390-396.

Berhe, S., Demurjian, S., Gokhale, S., Pavlich-Mariscal, J., & Saripalle, R. (2011). Leveraging UML for security engineering and enforcement in a collaboration on duty and adaptive workflow model that extends NIST RBAC. In Y. Li (Ed.), *Data and applications security and privacy XXV* (pp. 293–300). doi:10.1007/978-3-642-22348-8_25

Bodenreider, O. (2004). The unified medical language system (UMLS): Integrating biomedical terminology. *Journal Nucleic Acids Research*, *32*(1), 267–270. doi:10.1093/nar/gkh061

Boehm, B. (1986). A spiral model of software development and enhancement. *ACM SIGSOFT Software Engineering Notes*, *11*(4), 14–24. doi:10.1145/12944.12948

Booch, G., Rumbaigh, J., & Jacobson, I. (2005). *The unified modeling language user guide* (2nd ed.). Addison-Wesley Professional.

Brachman, R. J., & Schmolze, J. (1985). An overview of the KL-ONE knowledge representation system. *Journal of Cognitive Science*, *9*, 171–216. doi:10.1207/s15516709cog0902_1

Centricity, G. E. (2012). *GE centricity*. Retrieved from http://www3.gehealthcare.com/en/Products/Categories/Healthcare_IT/Electronic_Medical_Records

Chen, P. (1976). The entity-relationship model: Toward a unified view of data. *ACM Transactions on Database Systems*, *1*(1), 9–36. doi:10.1145/320434.320440

Cimiano, P., Hotho, A., & Staab, S. (2005). Learning concept hierarchies from text corpora using formal concept analysis. *Journal of Artificial Intelligence Research*, *24*, 305–339.

Craig, L. (2003). *Agile and iterative development: A manager's guide* (1st ed.). Addison-Wesley Professional.

Fernandez-Lopez, A., Grüninger, M., & Fox, M. (1995). Methodology for the design and evaluation of ontologies. In *Proceeding of Workshop on Basic Ontological Issues in Knowledge Sharing.*

Fernández-Lopez, M., Gomez-Perez, A., & Juristo, N. (1997). Methontology: From ontological art towards ontological engineering. In *Proceedings of the AAAI Spring Symposium* (pp. 33-40).

Fuentes-Fernández, L., & Vallecillo-Moreno, A. (2004). An introduction to UML profiles. *European Journal for the Informatics Professional*, *5*(2).

Genesereth, M. (1991). Knowledge interchange format. In *Proceedings of the 2nd International Conference on Priciples of Knowledge Representation and Reasoning* (pp. 238-249). Morgan Kaufman.

Glimm, B., Rudolph, S., & Völker, J. (2010). Integrated metamodeling and diagnosis in OWL 2. In *Proceeding of the 9th International Semantic Web Conference.*

Gomez-Porez, A. (1996). A framework to verify knowledge sharing technology. *Expert Systems with Applications*, *11*(4), 519–529. doi:10.1016/S0957-4174(96)00067-X

Gomez-Porez, A., Juristo, N., & Pazos, J. (1995). Evaluation and assessment of knowledge sharing technology. In N. J. Mars (Ed.), *Towards very large knowledge bases* (pp. 289–296). IOS Press.

Gruber, R. T. (2005). Toward principles for design of ontologies used for knowledge sharing. *Journal of Human Computer Studies*, *43*, 900–928.

Grüninger, M., & Fox, M. (1995). Methodology for the design and evaluation of ontologies. In *Proceeding of Workshop on Basic Ontological Issues in Knowledge Sharing.*

Guide, O. W. L. (2004). *OWL web ontology language*. Retrieved from http://www.w3.org/TR/owl-ref/#Header

Guranio, N. (2001). Formal ontology and information systems. In *Proceedings of 1st International Conference on Foraml Ontology and Infomration System,* Trento, Itlay.

Haase, P., Rudolph, S., Wang, Y., & Brockmans, S. (2005). *NeOn- Lifecycle support for networked ontologies*. Retrieved from http://www.neon-project.org/

Harold, E., & Scott, M. (2004). *XML in an nutshell*. O'Reilly Media.

Hartmann, J., Palma, R., & Sure, Y. (2005). OMV–Ontology metadata vocabulary for the semantic web. In *Proceeding of International Workshop on Ontology Patterns for the Semantic Web.*

Horrocks, I. (2002). DAML+OIL: A description logic for the semantic web. *IEEE Computer Society on Data Engineering*, *25*, 4–9.

Horrocks, I., Sattler, U., & Tobies, S. (1999). Practical reasoning for expressive description logics. In *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning* (pp. 161-180).

Horrocks, I., Sattler, U., & Tobies, S. (1999). Practical reasoning for expressive description logics. In *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning* (pp. 161-180).

ICD. (2009). *International classification of diseases*. Retrieved from http://www.who.int/classifications/icd/en/

Kuhn, M. (2010). Modeling vs encoding for semantic web. *Journal of Semantic Web-Interoperability, Usability. Applicability*, *1*(1), 11–15.

Lacy, L. (2005). *OWL: Representing Information using the web ontology language*. Trafford Publishing.

Liu, S., Ma, W., Moore, R., Ganesan, V., & Nelson, S. (2005). RxNorm: Prescription for electronic drug information. *IEEE IT Professional*, *7*(5), 17–23. doi:10.1109/MITP.2005.122

LOINC. (2001). *Logical observation identifiers names and codes*. Retrieved from http://loinc.org/

Microsoft Health Vault. (2013). *Microsoft health vault*. Retrieved from http://www.microsoft.com/enus/

Motik, B. (2005). On properties of metamodeling in OWL. In *Proceeding Of the 4th International Semantic Web Conference* (pp. 548-562).

Motik, B. (2005). On properties of metamodeling in OWL. In *Proceeding Of the 4th International Semantic Web Conference* (pp. 548-562).

Motik, B., Patel-Schneider, P., & Grau, B. C. (2009). *OWL 2 web ontology language direct semantics*. Retrieved from http://www.w3.org/2007/OWL/wiki/Direct_Semantics

Nicola, A., Missikoff, M., & Navigli, R. (2005). A proposal for a unified process for ontology building: UPON. In *Proceeding of 16th International Conference on Database and Expert Systems Applications.*

ODM. (2009). *Ontology definition metamodel (ODM)*. Retrieved from http://www.omg.org/spec/ODM/

OMG. (2011, August). *Meta object facility (MOF)*. Retrieved from http://www.omg.org/spec/MOF/2.4.1/PDF/

OWL 2 Profile. (2012, December 11). *OWL 2 web ontology language profile*. Retrieved from http://www.w3.org/TR/owl2-profiles/

Palmer, S. R., & Felsing, J. M. (2002). *A practical guide to feature-driven development* (1st ed.). Prentice Hall.

Pavlich-Marsical, P., Demurjian, S., & Michel, D. L. (2010). A framework of composable security features: Preserving separation of security concerns from models to code. *Special Issue on Software Engineering for Secure Systems*, *29*(3), 350–379.

Powers, S. (2003). *Practical RDF.* O' Reilly Media.

Protege. (2012). *Protege ontology editor*. Retrieved from www.protege.stanford.edu

Saripalle, R., & Demurjian, S. (2012). Towards a hybrid ontology design and development life cycle. In Proceeding of Semantic Web and Web Services. Las Vegas.

Saripalle, R., Demurjian, S., & Behre, S. (2011). Towards software design process for ontologies. In *Proceeding of 1st International Conference on Software and Intelligent Information.*

Uschold, M. (1998). The enterprise ontology. *Journal of The Knowledge Engineering Review*, 31-89.

Uschold, M., & King, M. (1996). Building ontologies: Towards a unified methodology. In *Proceeding of the 16th Annual Conference of the British Computer Society Specialist Group on Expert Systems.*

Winston, R. (1970). Managing the development of large software systems. In *Proceedings of the 9th international conference on Software Engineering* (pp. 328-338). IEEE Computer Society.

ENDNOTES

[1] A medical term denoting the priority of patients treatments based on the severity of their condition.

Rishi Kanth Saripalle is a final year Ph.D. student in the Department of Computer Science & Engineering at the University of Connecticut, under the supervision of Dr. Steven A. Demurjian. His research interests include: software engineering and modeling using various domain modeling standards, ontology design and development, knowledge engineering and modeling, software engineering applied to biomedical informatics. His research focuses on imposing software engineering, modeling and life cycle modeling concepts onto the domain ontologies, with the end purpose of defining a software engineering approach to designing and developing ontologies. He received his Masters, with a major in Computer Engineering, from the University Massachusetts, where he had research experiences in developing ontologies for aiding medical decision support system.

Steven A. Demurjian is a Full Professor and Director of Graduate Studies in Computer Science & Engineering at the University of Connecticut, and co-Director of Research Informatics for the Biomedical Informatics Division, with research interests of: collaborative security and access control models for role-based, mandatory, and discretionary approaches with security assurance for UML, XML, and cloud computing; biomedical informatics and software architectures for health information exchange; secure software engineering with UML; and, ontology design and development models and methodologies. Dr. Steven A. Demurjian has 150 archival publications, in the following categories: 1 book, 2 edited collections, 50 journal articles and book chapters, and 98 refereed conference/workshop articles.

Alberto De la Rosa Algarín is a Ph.D. student of Computer Science & Engineering at the University of Connecticut, with research interests including: information and knowledge-level security and privacy enforcement, document-level information security, identity-inferred access control, knowledge modeling and ontology engineering. He holds two majors, one in Computer Science and another in Mathematics, from the University of Puerto Rico.

Michael Blechner is an Assistant Professor of Pathology and Laboratory Medicine and Director of Pathology Informatics and Transfusion Medicine at UCHC, and a faculty member of the Biomedical Informatics Division, with medical informatics fellowship training as a National Library of Medicine (NLM) funded fellow in 2006. He is a skilled educator with teaching experience in both clinical and technology settings. Dr. Blechner's research interests include computerized decision support for laboratory medicine, data warehousing and optimization of clinical laboratory data for research and patient safety initiatives, and intelligent tutoring systems for medical training, especially in the context of laboratory utilization and test interpretation.