Chapter 10
Attaining Semantic Enterprise Interoperability through Ontology Architectural Patterns

Rishi Kanth Saripalle
University of Connecticut, USA

Steven A. Demurjian
University of Connecticut, USA

ABSTRACT

Enterprise Interoperability Science Base (EISB) represents the wide range of interoperability techniques that allow the creation of a new enterprise application by utilizing technologies with varied data formats and different paradigms. Even if one is able to bridge across these formats and paradigms to interoperate a new application, one crucial consideration is the semantic interoperability to insure that similar data is reconciled that might be stored differently from a semantic perspective. In support of this requirement, usage of ontologies is gaining increasing attention as they capture shareable domain knowledge semantics. The design and deployment of an ontology for any system is very specific, created in isolation to suit the specific needs with limited reuse in the same domain. The broad proliferation of ontologies for different systems, which, while similar in content, are often semantically different, can significantly inhibit the information exchange across enterprise systems. This situation is attributed, in part, to a lack of a software-engineering-based approach for ontologies; an ontology is often designed and built using domain data, while software design involves abstract modeling concepts that promote abstraction, reusability, interoperability, etc. The intent in this chapter is to define ontologies by leveraging software design pattern concepts to more effectively design ontologies. To support this, the chapter proposes Ontology Architectural Patterns (OAPs), which are higher-level abstract reusable templates with well-defined structures and semantics to conceptualize modular ontology models at the domain model level. OAP borrows from software design patterns inheriting their key characteristics for supporting enterprise semantic ontology interoperability.

DOI: 10.4018/978-1-4666-5142-5.ch010

1. INTRODUCTION

In today's world, the design, development, and deployment of a new enterprise application is no longer taking the prior approach of developing the application from scratch; rather, the emphasis is on the ability to construct a new enterprise application through the usage of existing resources such as enterprise applications, systems, servers, databases, etc., that are brought together to yield a system of systems. Enterprise Interoperability Science Base (EISB, Popplewell et al., 2012) has been promoted in order to address all of the different interoperability concerns including data, process, knowledge, cloud and Web services, rules, objects, APIs, etc. Two related interoperability of issues of particular interest are the ability to deal with: data in varied formats (e.g., XML, JSON, RDF (Allemang & Hendler, 2011), relational database, etc.) and the need to resolve semantics among enterprise systems of data (e.g., in a geospatial application, grid north vs. true north vs. magnetic north and these must be resolved if different do not use consistent formats). Ontologies have emerged to play a pivotal role in the World Wide Web (WWW) to promote the Semantic Web (Allemang & Hendler, 2011) by attaching semantics to electronically represented information thereby assisting users (humans and agents) in various ways such as semantic Web agents, semantic information extraction, semantic search, etc. Currently, ontologies are highly employed in the wide variety of enterprise applications for knowledge representation and reasoning (Baader, McGuinness, Nardi, & Patel-Schneider, 2007), software modeling and development (Demurjian, Saripalle, & Behre, 2009; Kuhn, 2010; Saripalle, Demurjian, & Behre, 2011), semantic information extraction (Wimalasuriya & Dou, 2010), biomedical and clinical informatics (Smith & Ceusters, 2006), databases (Gali, Chen, Claypool, & Uceda-sosa, 2004), geospatial semantics (Janowicz, Scheider, Pehle, & Hart, 2012), etc.

The primary goal of the ontologies is to capture semantics of a domain and tag the semantic concepts to electronically represented information, which in turn will ease *semantic interoperability* for enterprise applications to support both data and knowledge interoperability in EISB, assuming that the exchanging systems (e.g., computer systems, software applications, database records etc.) must come to an agreement on domain semantics in order to build an enterprise application. For example, various ontologies have been developed for capturing knowledge semantics on various aspects of a given domain for easing semantic interoperability issues in enterprise applications. For instance, in the business domain, the semantic Web has influenced various aspects of existing implementations such as: Simple Object Access Protocol (SOAP) (SOAP, 2007), Web Service Description Logic (WSDL) (WSDL, 2001), Service Oriented Architecture (SOA) (Bell, 2008), etc. In all of these approaches, the domain semantics captured in an ontology are tagged to business/service information represented using these standards, facilitating semantic compatibility between interacting enterprise services and easing knowledge interoperability (Nagarajan, Verma, Sheth, Miller, & Lathem, 2006; Burstein & McDermott, 2005). Researchers have also designed and implemented OWL-S (OWL-S, 2004), a semantic Web enabled Web-service model that incorporates all of the aspects of a software Web service lifecycle using ontology frameworks. For example, in the financial enterprise, lack of standard ontologies for capturing the semantics related to the financial domain have created a major bottleneck for information exchange/integration, knowledge extractions, financial reporting, Web services, etc., due to semantic ambiguity in the represented financial knowledge (Makela, Rommel, Uskonem, & Wan, 2007; Hu, 2010). Currently, Object Management Group (OMG) has taken an initiative to develop Financial Industry Business Ontology for capturing semantics related

to the financial domain (FIBO, 2012). As another example, in the government domain, semantic technologies such as linked data, semantic Web, ontologies, etc., have become a crucial component for achieving integrated e-government services (Bettahar, Moulin, & Barthes, 2009). These semantic components have been introduced into software architectures, providing semantics to electronically augment government information and facilitate semantic integration/interoperability between the participating government services/departments (Davis, Harris, Crichton, Shukla, & Gibbons, 2008; Fonou-Dombeu & Huisman, 2011), etc.

However, the success of employing ontologies for resolving enterprise semantic interoperability is jeopardized due to *structural* and *semantic interoperability* issues among the domain ontologies that are used for the systems that support a new enterprise application. There are a number of key issues to address. First, the individual ontologies of each constituent system used by a new enterprise 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 ontologist 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 enterprise 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 enterprise domain ant its appropriate set of ontologies in a more comprehensive and general manner. Such an approach towards ontologies is in direct conflict with the design methodologies in software engineering, databases, and Web settings, where the primary emphasis is on the modeling techniques that can applied to conceptualize the problem in a fashion that promotes characteristics such as modularity, abstraction and reuse, which implicitly eases structural and semantic interoperability issues. In this chapter, we leverage our previous work on extending the Web Ontology Language (OWL) for design and development of ontologies for applications that is more aligned with the software lifecycle and emphasizes a design approach for ontologies (Saripalle, Demurjian, & Behre, 2011; Saripalle & Demurjian, 2012b).

To provide a context for this chapter, we leverage an example in the healthcare domain, where it is necessary to construct an enterprise application for health information exchange (HIE) that is able to pull patient medical information from multiple sources in different formats and using alternative programming paradigms. An HIE enterprise application is constructed by gathering data from: electronic health records (EHRs) which repositories of patient medical records that may exist in provider offices, clinics, and hospitals; personal health records, (PHRs) that allow patients to manage their own health care data; personalized medicine health portals (PMHP) which allows providers to view their own patients' genetic data against their EHR in order to bridge the gap between providers and medical researchers; and, other laboratory, diagnostic, pharmaceutical systems that involve patient care. Note that an HIE enterprise application for many situations provides only read information to patient data or de-identified data sets. In support of an HIE enterprise application, the biomedical field provides

a significant variety of high-level XML standards including: the Continuity of Care Record (CCR) (ASTM, 2003), Continuity of Care Document (CCD), the Health Language Seven (HL7) Clinical Document Architecture (CDA) (HL7 CDA R2, 2008), etc. These high-level standards allowed medical providers to seamless *structure*, *integrate,* and *share* the patient's medical data with their respective propriety systems and collaborating environments. For providing enterprise semantic interoperability between the constituent systems (EHRs, PHRs, etc.), there are numerous standards such as: International Classification of Disease (ICD-10) (ICD, 2013), Logical Observation Identifiers Names and Codes (LOINC) (LOINC, 2013), Systematized Nomenclature of Medicine Clinical Terms (SNOMED-CT) (SNOMED CT, 2013), Diagnostic and Statistical Manual of Mental Disorder (DSM) (DSM, 2012), Unified Medical Language System (UMLS) (Bodenreider, 2004), etc. The problem that exists in the health care domain to hinder enterprise semantic interoperability are the inconsistencies in these low-level ontology standards, e.g., Psychoses (DSM, 2102) is a "Mental Disorder" in ICD and a "Psychotic illness" in DSM, while. Spherocytosis (SNOMED CT, 2013) is a "Diseases of Blood and Blood-Forming Organs" in ICD and "Red Blood Cell Shape-finding" in SNOMED-CT. These differences in the health care domain, along with similar cases in other enterprise domains, must be reconciled to achieve enterprise semantic interoperability.

In software engineering, a designer can better understand the domain problem and propose a plausible solution by developing domain model(s) to provide an abstract view of the solution with well-defined structure and semantics and are developed by considering domain instance data as it can influence the design, but not to the point that the design gets tied to the domain instances. One approach to software domain modeling that greatly facilitates reuse are software design patterns (SDP) defined as "a template illustrating a reusable solution to a reoccurring problem in multiple different situations with similar context" (Gamma, Helm, Johnson, & Vlissides, 1994; Freeman, Robson, Bates, & Sierra, 2004). SDPs can influence an enterprise application by allowing these generalized templates to be customized for a specific domain. For instance, the Model-View-Controller (MVC) pattern can be used easily in any enterprise application that requires: a *Model* that manages the behavior and data of the application; a *View* that manages the UI of the application; and, a *Controller* that interprets the user actions and informs the actions to the model and/or the view. Based on the domain application, the domain models replace the respective MVC components. This ability of SDP to divide the complex problem into modular manageable sub-problems and develop solutions that facilitate modularity, reusability, interoperability, etc., has gained them a prominent position in the software community; our intent is to extend SDPs to support enterprise semantic interoperability for ontologies.

In this chapter, the overall goal is to improve the design, development, and deployment of ontologies with syntactic and semantic integration in support of an enterprise application by proposing a set of *Ontology Architectural Patterns (OAPs)* which are abstract reusable architectural patterns influencing the overall development of semantic knowledge involving multiple ontology models that ease interoperability issues and promote a software engineering approach for ontology development. This have been introduced to a limited extent in our prior work on semantic patterns (Saripalle & Demurjian, 2012a), but this chapter dramatically extends that work by leveraging SDP concepts to ease *structural* and *semantic interoperability* issues. In the process, the ontologist is encouraged and guided to design and develop modular ontology models, reusable in multiple domain application settings that share similar con-

textual requirements. As a result, the ontologist can focus on constructing reusable ontology models, moving future reconciliation and integration of different ontologies for an enterprise application from the current labor-intensive instance level to a more abstract and conceptual domain model level. The OAP extensions are illustrated using an HIE enterprise application that brings together different systems (EHRs, PHRs, etc.) in support access across multiple sources for two types of users: medical providers that are interested in obtaining a full picture of a patient's medical data collected from varied sources to facilitate clinical care; and, researchers that seek to analyze data from the HIE Repository that has been de-identified to conduct disease related, public health surveillance, and other research. HIE is a complex enterprise application that must marshal data from multiple sources and can be greatly benefited by OAPs to model a solution that can resolve and reconcile syntactic and semantic differences among the ontologies of its constituent systems.

Towards this goal, the reminder of this chapter is divided into five sections. Section 2 provides additional background and motivation on the role of ontologies in the design and development process, a review of software design patterns, their classification, and usage in the software development process, and a brief introduction to enterprise interoperability (EI) and the role that ontologies can play, particularly in regards to our proposed OAPs. Section 3 proposes the developed Ontology Architectural Patterns (OAP) which are explained in terms of name, motivation, description, context & usage, and application & implementation; examples in both health care and other enterprise domains are provided Section 4 reviews related research in ontology patterns and ontology frameworks, comparing and contrasting to our work presented in Section 3. Section 5 discusses the future direction of knowledge development for facilitating enterprise interoperability. Finally, the last section concludes the chapter.

2. ONTOLOGIES, SOFTWARE DESIGN PATTERNS, AND ENTERPRISE INTEROPERABILITY

This section provides additional explanation on motivation and background for the paper. First, Section 2.1 reviews additional motivation on ontologies regarding the role that they can plan in systems design and development, with an example in the health care domain that further motivates the need for Ontology Architectural Patterns (OAP). Using this as a basis, Section 2.2 briefly examines Software Design Patterns (SDP) in terms of their inception, categories, and role in the software development process. Finally, Section 2.3 introduces the domain of Enterprise Interoperability (EI), its concepts and relationship with Sections 2.1 and 2.2, to place the work of this chapter into a proper perspective with EI.

2.1. Role of Ontologies in Systems Design and Development

The discussion in Section 1 provided an initial motivation for the need and usage of Ontology Architectural Patterns (OAP) for enterprise interoperability through a discussion of the status of ontologies, design, and modeling approaches of knowledge and software engineering and software design patterns. This section further clarify the motivation to precisely position the work on OAP with respect to traditional ontology development and its need for a higher level abstract approach akin to software design patterns. The end result provides a means to more precisely and generally define ontologies (and their components) with an aim towards potential reusability in a domain (or program family).

Current approaches to ontology development today are conducted using tools such as Protégé (Protege, 2012), Java Ontology Editor (JOE) (Java Ontology Editor, 1998), etc. These tools allow an ontology developer to work with domain

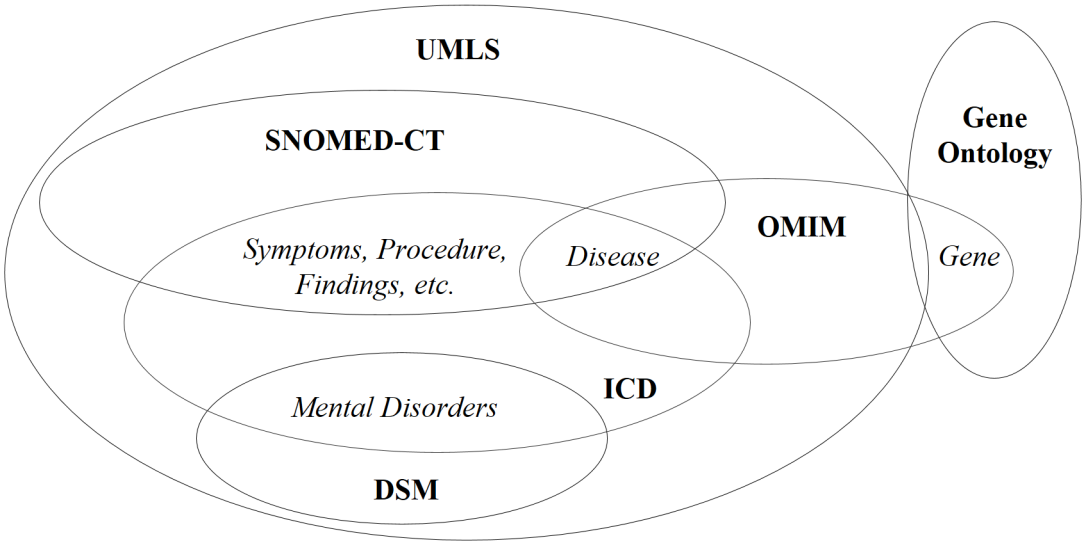
experts with a focus on building an ontology that maximizes the domain knowledge by capturing the knowledge concepts for a given domain of disclosure. This leads to a current ontology development practice that is predominantly *instance* and *construction* based (Kuhn, 2010; Saripalle, Demurjian, & Behre, 2011; Saripalle & Demurjian, 2012b), totally focused on the application and its specific domain and system requirements. Such a process is highly influenced by the ontology developer's experience, input from domain experts, and based on the actual data that is to be modeled within the ontology rather than defining a structure for the ontology that is more reusable across enterprise systems. As a result, ontologies created for different systems of the same domain can be quite different. For example, constructing a set of ontologies for enterprise application such as Amazon would require a very broad set of ontologies to capture all of its products characteristics (books, electronics, household, fashion, etc.), while a set for Barnes & Noble may be limited much more to books. The two resulting sets of ontologies developed for Amazon and Barnes & Noble may be difficult to integrate, with human intervention need to understand equivalences across the two sets, a laborious task with at best semi-automatic methodologies and reuse of the ontology is obscure. The OAP and an emphasis on that ontology design would allow a more generalized ontology for "books" to be created that would have the potential to be utilized in all enterprise applications where book sales are a part of their business process.

Second, ontologies cannot be created in a vacuum without consideration of concepts that may cut across different solutions (different enterprise applications) and require sophisticated modeling to abstract domain specific materials out of the ontology to yield an ontology schema (akin to an XML schema) that is reusable across multiple enterprise applications of a given domain. Further, a single ontology may not suffice for the structural and knowledge requirements of the enterprise application, but may require multiple interacting ontologies to correctly represent the required medical knowledge. Continuing with the earlier example, there may be an ontology architectural pattern that can be applied at a higher level across the Amazon and Barnes & Noble enterprise applications that is able to capture ontology similarities at a higher abstraction level that promotes reuse. In the health care domain, there could be many enterprise applications that would need to utilize the same ontology. For example, a clinical researcher trying to identify genes responsible for mental disorders might want to have an enterprise application to access genomic and clinical (patient) databases that would require the integration of the Gene Ontology (Ashburne & Lewis, 2002) and the Diagnostic and Statistical Manual of Mental Disorders (DSM) to develop a Gene-Mental Disorder ontology. Likewise, another enterprise application could provide the ability to allow queries to cross the International Classification of Diseases (ICD-10) codes with Logical Observation Identifiers Names and Codes (LOINC) ontology to develop a Disease-Laboratory Test correlation ontology to be used by combining ICD-10 and LOINC with de-identified patient data for mining. However, the reconciliation process between these ontologies will be arduous, performed on the instance level rather than at a higher-level of abstraction that would involve the structure of the ontologies. Even, if an expert ontologist develops a massive single ontology by integrating the required multiple ontologies, the large-scale nature of this massive ontology (defined and then integrated at an instance level) sacrifices fundamental software characteristics such as modularity, reusability, abstraction, minimal coupling, etc. The integration/mapping rules between these participating ontologies are contextually based on an organization application goals, meaning that their integration may not be easily accomplished if their purposes are so diverse as to make it difficult to identify linkages and commonalities. We note again that the current ontology integration methodologies

focus on integrating ontology instances using linguistic and statistical techniques that often ignore semantics. Such an approach can result in a merging that cannot be syntactically and semantically verified. This is evident from studies on the UMLS (Krauthammer, 2002; Kohler, 2007) and SNOMED-CT (Chiang, Hwang, Yu, et al., 2006) medical standards where they have been attempts to integrate two ontologies that have significantly different nomenclature. This is true for many other domains.

Finally, while the existing medical ontologies previously discussed have an overwhelming knowledge overlap among them, the way that each ontology deals with that knowledge can be quite different. Further, since these ontologies have often been built in isolation (e.g., by different standard committees and other organizations), having an ontologist that can understand all of them is unrealistic. For example, if there are one or two features a newly developed ontology needs that are not in the standard, the solution to develop yet another ontology occurs, often replicating information from multiple ontologies and having a vocabulary that may be in conflict with other standards the "same" information (semantically). Thus, the existing ontologies, the newly development ontologies, and any future customized ontologies have to be constantly monitored and integrated with one another in order to remain consistent. This is a monumental task in any enterprise application, further compounding the structural and semantic interoperability issues with ontologies. Imagine the difficulty it would be to get the organizations (and their enterprise applications) for Amazon, Barnes & Noble, eBay, etc., to all work towards a common shared ontology. Obtaining agreement from such a large number of stakeholders will be difficult to achieve in practice. Figure 1 supports this argument by illustrating the domain knowledge overlap between medical ontologies. Notice in Figure 1 that the OMIM and Gene Ontologies have overlapping knowledge on Gene domain. In addition ICD and DSM have a knowledge overlap on the domain of Mental Disorders, with SNOMED-CT and ICD having a major knowledge overlap on multiple domains such as Disease, Symptom, Procedure, etc. Further, SNOMED-CT, ICD and OMIM have a knowledge overlap on domain of Disease. More significantly, UMLS is attempting to encompass all of the standards under one umbrella via its own theory and model

Figure 1. Domain knowledge overlap between various standard medical ontologies



for achieving its goal, but fails to provide modular domain models for respective domains which can reused independent of the domain application. Thus, failing to reuse existing knowledge sources and developing new ontologies for targeting the same domain knowledge leads to chaotic structural and semantic interoperability issues.

2.2. Software Design Patterns

In software engineering, Software Design Patterns (SDP) arose when developers noticed that they were using the same software structure in terms of classes, interfaces, and interactions in multiple settings, tweaking them to handle difference in domain data. SDPs expand the concept of generics (used to capture a lower level component like a stack that can be instantiated for any data type) to a higher-level pattern that captures the generalized structure, semantics, direction and usage at of a set of classes, interfaces, and their interactions that represents a major component of a system. As a result, SDPs are adaptable to work in varied settings for enterprise applications that have similar design requirements or behavioral characteristics. The primary components of a SDP are: *context* that explains when the design pattern is applicable by defining environment parameters and the usage of the pattern itself; *problem* that illustrates the kind of problems the pattern can applied to; and, *solution* to explain the way to use the pattern as a viable software engineering solution to the encountered domain problem. Apart from these components, a design pattern also may have the following essential elements: *name* of the pattern; examples to illustrate the application of the pattern to a multiple domain specific problems; *rationale* for the logic explanation of the pattern and its application; *related patterns* that have the same or varying categories or types; and *known uses* for successful industrial usecase scenarios. SDPs have gained importance since they are generalized artifacts from multiple solutions they are domain independent making them interoperable between heterogeneous domain application problems. The use of SDPs in Web-based solutions are foundational regardless of the implementation platform.

SDPs are classified into three broad categories that are based on the functionality, interactions, and purpose of the SDP (Freeman, Robson, Bates, & Sierra, 2004). The first category, the *Creational SDP,* deals with a software entity's (mostly classes) creation in a manner suitable to the given application context. Sample creational SDPs include: the Abstract Factory Pattern that provides a way to encapsulate a group of individual factories that have a common theme without specifying their actual classes; the Factory Method pattern that defines the interface for creating an object, but allows subclasses to decide which class to instantiate; and, the Builder Pattern that separates the concerns of construction of a complex object from its representation,. The second category, the *Behavioral SDP*, is used to identify the common communication patterns between objects. Sample behavioral SDPs include: the Chain of Responsibility Pattern consisting of a source of command objects and a series of processing objects where each processing object contains logic that defines the types of command objects that it can handle; the Command Pattern in which an object is used to represent and encapsulate all of the information needed to call a method at a later time; and, the Observer Pattern where an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods. The third category, *Structural SDP*, provides a simple way to realize relationships between multiple entities. Sample structural SDPs include: the Adapter Pattern that allows classes to work together that normally could not because of incompatible interfaces; the Bridge Pattern that decouples an abstraction from its implementation so that the two can vary independently; and, the Composite Pattern which describes that a group of objects are to be treated in the same way as a single instance of an object, with the intent to

compose objects into a tree structure to represent part-whole hierarchies. SDPs play a significant role in enterprise applications; our intent is to expand this concept to include knowledge modeling which will allow the specification of ontology architectural patterns (OAP) that can play a major role in achieving enterprise semantic interoperability.

2.3. Enterprise Interoperability (EI)

Enterprise Interoperability (EI) (Charalabidis, Goncalves, & Popplewell, 2010; Jardim-Goncalves, Grilo, Agostinho et al., 2013) is defined

… as a field of activity with the aim to improve the manner in which enterprises, by means of information and communications technologies, interoperate with other enterprises, organizations, or with other business units, in order to conduct their business (Popplewell et al., 2012).

The reality is that new enterprise applications are built today by the cobbling together of functionality from multiple sources (via APIs, Web services, cloud services, JSON calls, etc.) and their interoperation requires addressing different facets associated with enterprise development. The overriding object of EI is to support the ability of enterprises to communicate and interact seamlessly. The term Enterprise can be defined as "an organization that provides open/paid services to clients" (Charalabidis, Goncalves, & Popplewell, 2010) and Interoperability can be defined as

…the ability of two or more systems or components to exchange information and to use the information that has been exchanged (Popplewell et al., 2012).

EI as a generalized concept, subsumes other interoperability issues from differing perspectives, detailed in ESIB Report (Popplewell et al., 2012): Data Interoperability, Process Interoperability, Knowledge Interoperability, Services Interoperability, Rules Interoperability, Objects Interoperability, Software Interoperability, Cultural Interoperability, Social Networks Interoperability, Electronic Identity Interoperability, Cloud Interoperability, and Ecosystems Interoperability. These noted interoperability issues are interweaved with one another at a conceptual level. For example, there is a strong dependency between Data and Knowledge Interoperability, Process and Knowledge Interoperability, and Process and Service Interoperability, to name a few. When one addresses the issues for a given interoperability, there is a corresponding ripple effect on other interoperability issues.

The proposed ontology architectural patterns (OAP) in this chapter has the primary goal to solve semantic interoperability among domain ontologies and is primarily targeting Knowledge Interoperability and is closely related to Data Interoperability. Hence, providing a software-engineering-based solution to semantic interoperability among ontologies has an impact on the Knowledge and Data Interoperabilities of EI. Additionally, in support of OAP, we leverage the related works of Gangemi's (Gangemi, 2005; Gangemi & Presutti, 2009) proposed Conceptual Ontology Design Pattern (CODeP), Clarks's (Clark, Thompson, & Porter, 2004) abstract Knowledge Patterns (see Section 4) for designing more effective ontology models at a high abstraction level via ontology semantic patterns. For example, multiple enterprise systems can employ a Time-Indexed Participation CODeP for defining their own open-source/proprietary ontology model for online services, where: Object in the CODeP pattern can represent physical items, online Web services, person/semantic agent, etc.; Event can be cast as service orders, automated triggers, internal events, etc.; and, Time Interval can represent a time frame. Even though the service ontology model(s) and respective implementation vary between the enterprises, the ontology models that are utilized refer to the same semantic pattern (including the semantic context). This results in

reusing the patterns concepts (classes, attributes and associations) and application context thereby easing Knowledge Interoperability when enterprises need to collaborate.

3. ONTOLOGY ARCHITECTURAL PATTERNS (OAP)

The emergence of the eXtensible markup language (XML) as a near defacto standard for information representation and exchange has had a significant impact on the ontology and enterprise interoperability research and development areas. XML dominates standards in computing and other fields, in addition to all of the aforementioned healthcare standards (HL7 CDA, CCR, etc.) there are many other standards that have the potential to impact enterprise applications including: HR-XML (HR-XML, 2013) for personnel and developers to have a common terminology for all aspects of human resources; the Oasis Open Office XML (Brauer & Schubert, 2013) document format for representing documents, presentations, etc., more easily; a wide variety of standards for libraries (LOC, 2012) such as the METS standard for tracking metadata on objects in digital collections; the Oasis LegalXML (LegalXML, 2008) standard for the electronic exchange of legal data and documents; and so on. In addition, ontologies can capture and attach semantic knowledge to represented information thereby aiding users (humans and agents) in knowledge engineering and representation, domain modeling, database and object-oriented analysis, natural language processing, biomedical and clinical informatics, etc. These efforts are supported by a wide variety of knowledge representation frameworks such as Resource Description Framework (RDF) (Powers, 2003), Web Ontology Language (OWL) (OWL Guide, 2004; Lacy, 2005), KIF (Genesereth, 1991), DAML+OIL (Horrocks, 2002), etc. Clearly, all of these standards and frameworks are available for a wide range of enterprise applications, and will be more and more important over time.

In this section, the work on ontologies as discussed in Section 2.1 and issues related to enterprise interoperability in Section 2.3 provides a strong justification to extend and apply software design pattern concepts so that they are suitable for ontology design. Specifically, this section details our developed *Ontology Architectural Patterns* (OAP) defined as abstract reusable architectural patterns that can assist the domain designer to define reusable modular ontology models at a higher abstraction level in order to support enterprise interoperability. The primary aim of developing OAP is twofold. First, OAP as a modeling construct eases the ontology architectural design process by providing the ability to define a more general solution for a domain application that attains required domain knowledge. Second, these OAPs promote development of modular domain ontology models encouraging knowledge abstraction, minimizing coupling, facilitating reuse, etc., which has the end result of easing *interoperability* issues between ontologies thereby facilitating enterprise interoperability of data and knowledge. In order to achieve these goals, we have developed three OAPs. The *Linear Ontology Architectural Pattern (LOAP)* is presented in Section 3.1 and represents a linear/parallel architectural arrangement of multiple ontology models for achieving the required knowledge goal in a manner where the captured knowledge is accessible in an ordered manner. Then, in Section 3.2, the *Centralized Ontology Architectural Pattern (COAP)* is explained, defining a global ontology model (higher-level abstraction) under which multiple component ontology models interact akin to a centralized hub. Lastly, in Section 3.3, the *Layered Ontology Architectural Pattern (LaOAP)* in defined to support a layered arrangement of the components organized from the innermost to outermost layer as: the Ontology Conceptual Model (innermost layer) within the Axiom and Rule for the Model within the

Mapping of the Model within the Terminology (vocabulary) of the Model within the Query and Web Services (outmost layer).

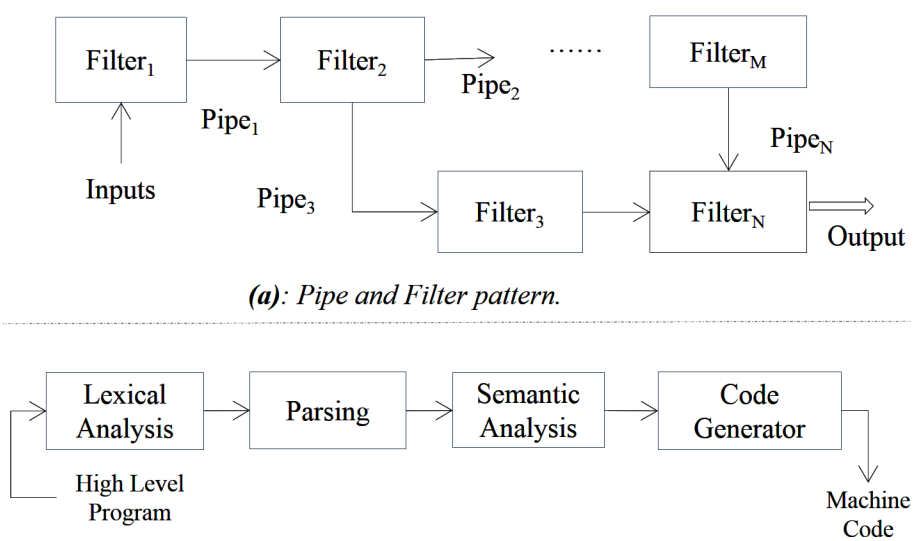
To standardize the discussion in the remainder of this section, the presentation follows a consistent ordering for each *Ontology Architectural Patterns* (OAP) expressed with: the *Pattern Name* which identifies the OAP by name; the *Pattern Motivation* which is utilized to more fully motivate the need of the proposed pattern and the influence of various SDP, ontologies, and their frameworks from other domains; the *Pattern Description* which explains the operational and functional interactions of each pattern; the *Pattern Context & Usage* which represents the contextual requirements for applying the pattern and its usage in an enterprise application; and, the *Pattern Application & Implementation* which illustrates an instantiation of the pattern and its realization in an enterprise application. Throughout the discussion, we provide examples using the domains of biomedical informatics, e-commerce, e-services, e-government etc.; again, we note that these patterns are general and can be applied to any domain.

3.1. Linear Ontology Architectural Pattern

Pattern Name: Linear Ontology Architectural Pattern (LOAP).

Pattern Motivation: The *Linear Ontology Architectural Pattern (LOAP)* is primarily motivated and influenced by the Pipe & Filter and Chain of Responsibility SDPs that are categorized under structural SDPs as defined in Section 2.2. The primary goal of these patterns is to divide the large complex problem into smaller modular problems and develop generic reusable solutions. The *Pipe & Filter SDP* as shown in Figure 2a has two primary components: Filter which holds the logical modules (e.g., file readers, boot loaders, memory units, etc.) that accept the given input(s), processes them, and generates an output; and, Pipe which interconnects two or more Filters (i.e., the output of one Filter is fed in as input to another Filter). For example, compliers might utilize Pipe & Filter SDP where the complex problem of compiling

Figure 2. The Pipe & Filter SDP and its implementation



***(a)**: Pipe and Filter pattern.*

***(b)**: Complier using Pipe and Filter pattern.*

a program is divided into smaller problem modules such as Lexical Analysis, Parsing, Semantic Analysis, and Code Generator. These modular components are connected and executed accordingly to obtain the final output that is machine executable code as shown in Figure 2b.

Similarly, the *Chain of Responsibility SDP*, as shown in Figure 3a, also has two primary components: Logical Handlers, which are generally governed by a superior interface (Handler Interface) to manage the application logic; and, the Relationship that handles the interactions between the participating Logical Handlers.

For example, as shown in Figure 3b. when a patient arrives at the clinic practice there may be several steps: nurse or medical assistant brings the patient to a room, reviews the medications, measures BP, pulse, temperature, etc., and notes on the purpose of the patient's visit; physician visits the patient, makes an assessment, and comes up with a treatment plan that may be a diagnosis (e.g., has strep throat take Augmentin) or require further evaluation (blood work and/or x-rays); the nurse or medical assistant may return to answer any follow-up questions and to provide appropriate prescriptions or other treatment instructions. In the example, all of the individuals involved in this process take up their responsibility (acting as Logical Handlers) and communicate (Relationship) the information to the next individual (nurse review→ physician assessment→ medical assistant/nurse action) to finally arrive at the outcome (successful treatment of the patient). The main difference between Chain of Responsibility and Pipe & Filer is for the latter to allow cyclical connections and bi-directional flow.

Pattern Description: The *Linear Ontology Architectural Pattern (LOAP)* as shown in Figure 4 is an architectural arrangement of ontology models ($Ontology_1$, $Onotology_2$,…, $Ontology_N$) which are connected ($Link_1$, $Link_2$,…,$Link_N$) in a sequential and/or in parallel order for achieving the

Figure 3. Chain of Responsibility SDP and its implementation

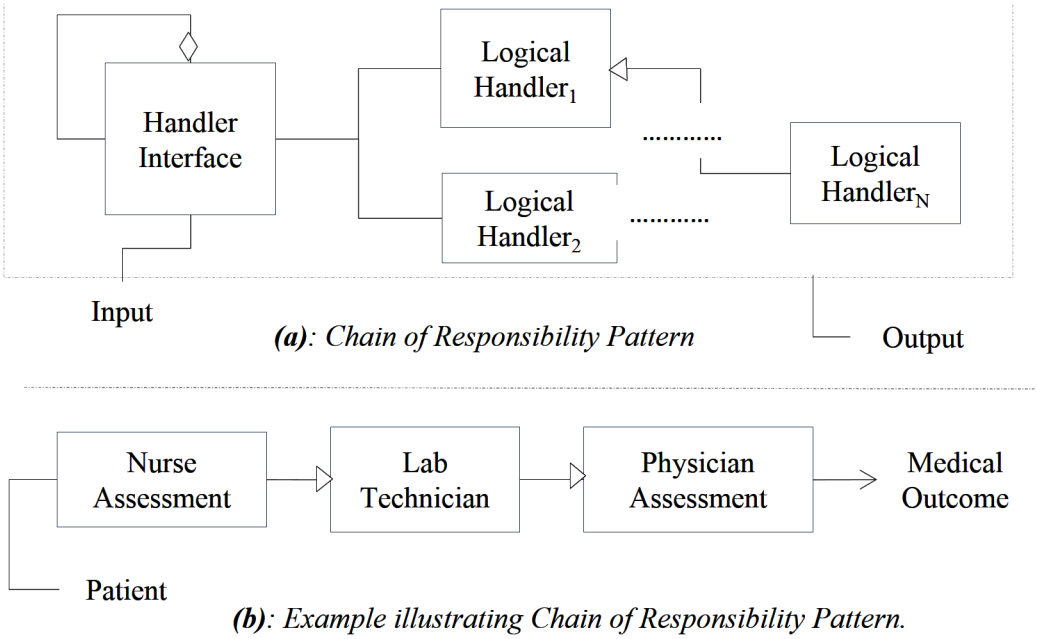
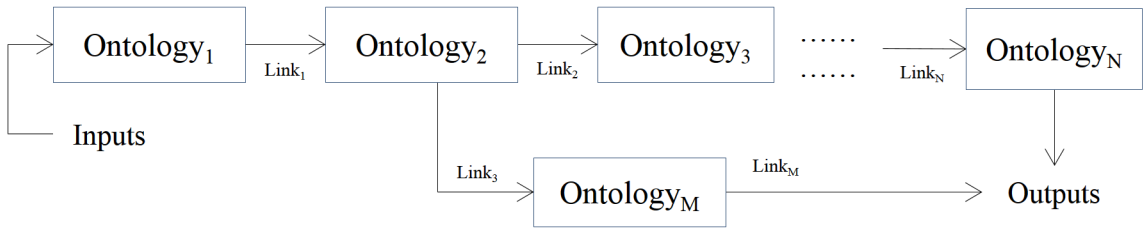


Figure 4. Linear Ontology Architectural Pattern (LOAP)



desired knowledge goal. The LOAP is developed by combining Pipe & Filter and Chain of Responsibility SDPs, where the ontology models (Ontology$_1$, Onotology$_2$,…,Ontology$_N$) are aligned to Filters (Filter$_1$, Filter$_2$, …,Filter$_N$, Figure 2a) and Logical Handlers (Logical Handler$_1$,…,Logical Handler$_N$, Figure 3a), and the connecting links (Link$_1$, Link$_2$,…,Link$_N$) correspond to Pipes (Pipe$_1$, …,Pipe$_N$, Figure 2a) and Relations (Figure 3a). The output is primarily a query result performed on the multiple ontology models based on the inputs (initial inputs or previous ontological outputs). The connections (Link$_1$, Link$_2$, …Link$_N$) between the ontology models are generally unidirectional but can also be a loop within an ontology model or between ontology models, where the data flows from the previous ontology model (Ontology$_1$) to the next ontology model (Ontology$_2$) and so on until the final output is generated. The initial input is given to Ontology$_1$, then the output of Ontology$_1$ is fed as input to Ontology$_2$ through Link$_1$ and the chain continues as the result is finally obtained at Ontology$_N$ to generate an output. One step with LOAP can also generate intermediate results from an entire other OAP (i.e., for some M, Ontology$_M$). The desired knowledge goal is achieved by developing modular reusable ontology models to ease semantic interoperability issues during the ontology models integration/reconciliation process.

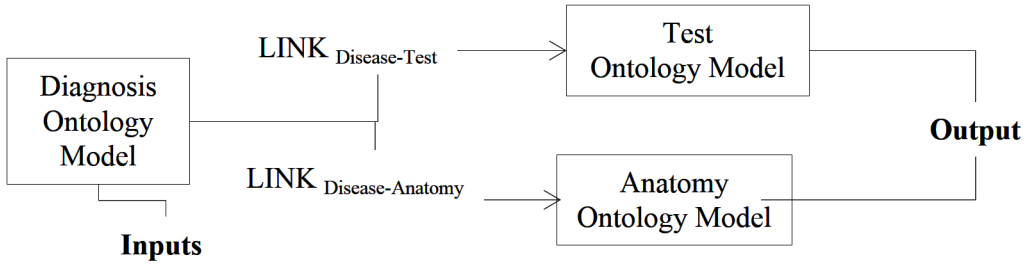
Pattern Context and Usage: The LOAP pattern is applicable to any enterprise domain application where the required domain knowledge is obtained by connecting multiple source ontology models or a single large ontology has to be divided into multiple modular ontology models that will then be connected in sequential/parallel fashion. The ability to separate a larger ontology into different logical component ontologies has an advantage in an enterprise interoperability context to allow the separation of various data and knowledge components. The links (Link$_1$, Link$_2$,...., Link$_N$ in Figure 4) between the ontology models is primarily *programmatic* consisting of the software logic that is written using languages such as Java, C++, etc., or *semantic* where the link semantics are captured in another ontology model. By designing and connecting reusable ontology models for capturing the required domain knowledge, the LOAP and its component ontology models are reusable in multiple enterprise application settings.

As an example, recall Figure 3 for the treatment of a patient in multiple steps. Underlying this process is the need to organize the information that is used into different ontology models that would be utilized in different steps of Figure 3. By analyzing the treatment process, an ontologist can design a Triage LOAP (pattern name) as shown in Figure 5a that has three Ontology Models (Diagnosis, Anatomy, and Test) with associated

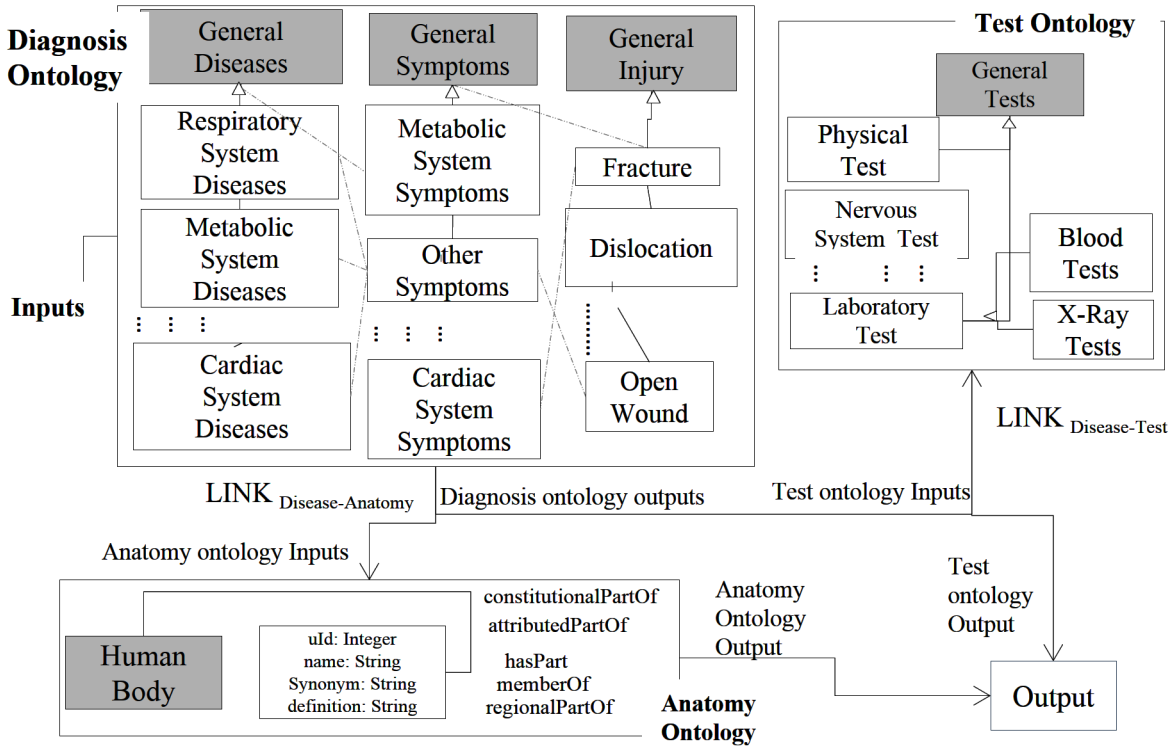
links ($Link_{Disease\text{-}Anatomy}$ and $Link_{Disease\text{-}Test}$). In Figure 5a, the Diagnosis Ontology model captures the knowledge on various Disease, Symptom, and Injury models, an Anatomy Ontology model captures the domain knowledge of the human body, and a Test Ontology model captures the knowledge on blood tests, imaging tests, cardiac tests, etc. Figure 5b expands each of the Ontology model blocks of Figure 5a demonstrating that within each step resides a significant ontology (that may also be an OAP). By modularizing the domain knowledge into multiple ontology models, the domain ontology models have the potential to be reusable in different enterprise applications, e.g., the Disease and Symptom Ontology models may only be needed for one enterprise application, with the Anatomy model used by another application.

Similarly, in the domain of e-government, there is constant need to exchange information between various departments such as public administration

Figure 5. Instance of a triage LOAP



(a). LOAP implementation using Diagnosis, Test and Anatomy Ontologies.



(b). Diagnosis, Test and Anatomy Ontology Models.

services, immigration services, tax department, immigration law bureau, etc. The ontology designer for the e-government domain can modularize the needed ontology models for maximizing the reuse of semantic knowledge among the government services. For example, a Public Service Ontology model captures the semantics (e.g., codes, description, definitions, eligibility, etc.) of the government public services, an Immigration Ontology model captures the semantics (e.g., types of visas, statuses of visa, etc.) of the government immigration domain, a Tax Ontology model captures knowledge about the tax codes of the government, and an Immigration Law Ontology model captures sematic knowledge about various laws involving immigration. Thus, based on the required knowledge goal, the e-government ontology models can be interconnected or linked (e.g., $\text{Link}_{\text{Tax-Immigration}}$, $\text{Link}_{\text{Service-Immigration}}$, $\text{Link}_{\text{Immigration-Law}}$, etc.) with one another similar to Figure 5a. Once the ontologies is developed for the e-government domain, subsets can then be applied to other enterprise applications for that domain.

Pattern Application and Implementation: The Triage LOAP in Figure 5 can be realized as shown in Figure 6 using a combination of: the OWL Framework (Lacy, 2005) for defining the three ontology models; the Protégé Ontology Editor (Protege, 2012) for building OWL based ontology models; the SPARQL (Lacy, 2005) query language to interrogate the Triage LOAP and its ontology models; and, Java for UI and program logic. The implementation of Figure 6 for an enterprise application as shown in Figure 7 adopts a three-layered approach where: Layer 1 holds the Java based UI for the user to enter the inputs, Layer 2 holds the logic of the application implemented using Java and SPARQL, and Layer 3 holds the actual OWL ontology instances for each model. The layered approach allows each layer to be independent and reusable from the other layers. The flow of the Triage LOAP (Figure 5) and its implementation (Figure 6) begins when the user (e.g., nurse, physician assistant, etc.) types in symptoms such as fever, cold, and fatigue (Layer 1) that are read using a Java program (Layer 2) and then fed into the SPARQL query engine (Layer 2). Next, the SPARQL engine feeds the user inputs to the Diagnosis Ontology Model whose outputs are given as inputs to both the Test and Anatomy Ontology Models.

The query performed on the Diagnosis Ontology model (LOAP_Diagnosis.owl, Figure 6a) by the SPARQL engine is shown in Figure 8a, which queries the model for known diseases for the given input symptoms. The links $\text{Link}_{\text{Disease-Test}}$ between the Diagnosis and Test Ontology Models (Figure 6) are implemented as a SPARQL query as shown in Figure 8b, which queries the Test Ontology Model (LOAP_Test.owl, Figure 6b) based on the Diagnosis Ontology Model outputs (d1, d2,…., dn). The $\text{Link}_{\text{Disease-Anatomy}}$ between the Diagnosis and Anatomy Ontology Models (Figure 6) is also implemented as a SPARQL query as shown in Figure 8c, which queries the Anatomy Ontology Model (LOAP_Anatomy.owl, Figure 6c) based on the Diagnosis Ontology Model outputs (d1, d2,…., dn).

3.2. Centralized Ontology Architectural Pattern

Pattern Name: Centralized Ontology Architectural Pattern (COAP).

Pattern Motivation: The *Centralized Ontology Architectural Pattern (COAP)* is influenced by the Façade SDP, the Local As View (LAV) methodology (Lenzerini, 2002), and the MAFRA framework (Maedche, Motik, Silva, & Volz, 2002). The Façade SDP provides a unified higher-level global interface/system developed from a set of complex heteroge-

Figure 6. OWL implementation of triage LOAP shown in Figure 5

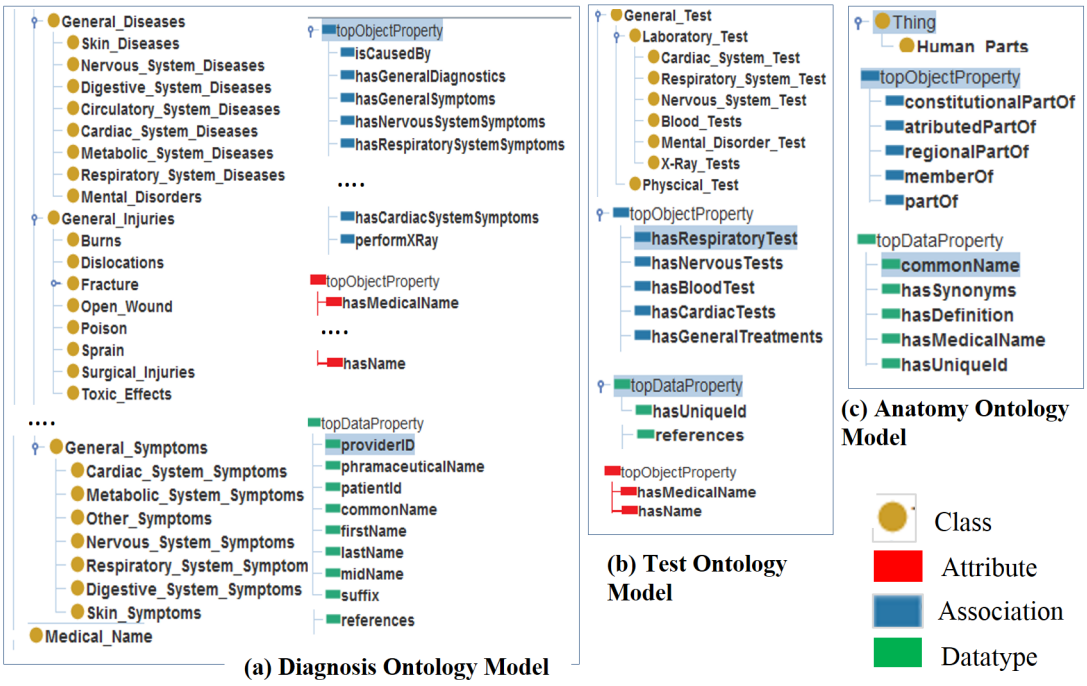


Figure 7. Layered implementation of triage LOAP shown in Figure 5

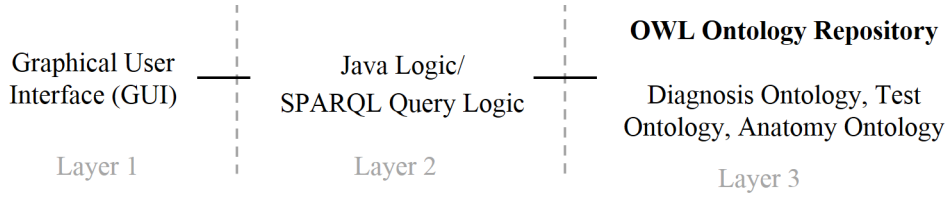


Figure 8. The SPARQL queries representing the Diagnosis Ontology Model query, $Link_{Disease\text{-}Anatomy}$, and $Link_{Disease\text{-}Test}$ between the ontology models as show in Figure 6

```
PREFIX LOAP:
<http://www.owl-ontologies.com
        /LOAP_Diagnosis.owl/>

SELECT ?commonName
WHERE {
    ?disease LOAP:hasSymptom "Fever".
    ?disease LOAP:hasSymptom "Cold".
    ?disease LOAP:hasSymptom "Fatigue".
}
```

(a): SPARQL Symptom query.

```
PREFIX LOAP:
<http://www.owl-ontologies.com
                /LOAP_Test.owl/>
SELECT ?commonName
WHERE {
    ?t LOAP:hasGeneralTreatments ?d1.
    ?t LOAP:hasGeneralTreatments ?d2.
    ….
    ?t LOAP:hasGeneralTreatments ?dn.
}
```

(b): SPARQL query representing the LinkDisease-Test.

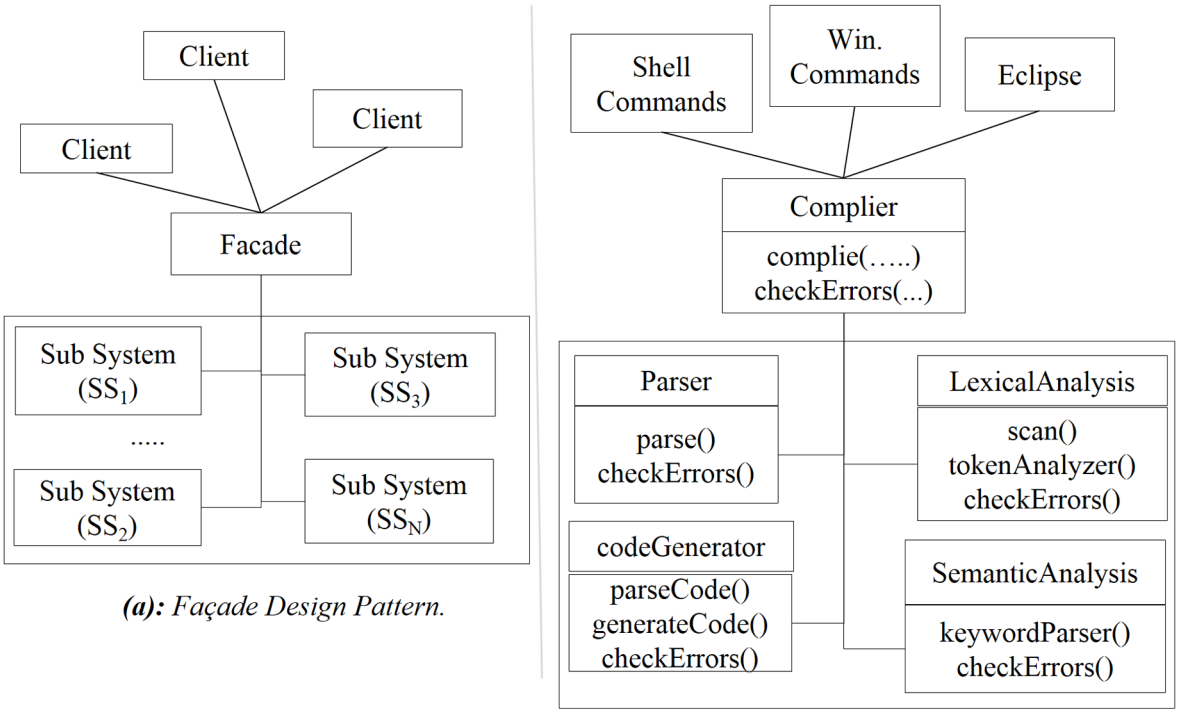
```
PREFIX LOAP:
<http://www.owl-ontologies.com/
    LOAP_Anatomy.owl/>
SELECT ?commonName
WHERE {
    ?a LOAP:effects ?d1.
    ?a LOAP:effects ?d1.
    ….
  ?a LOAP:effects ?dn.
}
```

(c): SPARQL query representing the LinkDisease-Anatomy.

neous source interfaces/subsystems making these local sources easier to utilize for the clients. As shown in Figure 9a, the Façade SDP abstracts common features or functional implementation from multiple complex subsystems (SS_1, SS_2… SS_N) and provides a unified simple global system (Façade) to access these complex subsystems for multiple clients, thus hiding the complexity of the subsystems. For example, the compiler example from the Pipe & Filter in Figure 2 can also be implemented using Façade as shown in Figure 9b, where the clients will call the simplified complier's functionality (function call - compile) which in turn performs the complex process of invoking other subsystems such as Lexical Analysis, Parser, SemanticAnalysis and codeGenerator in an appropriately defined order.

The LAV methodology is a data integration approach where a set of local schemas (LS_1, LS_2, LS_3… LS_N) are expressed as database views (V_1, $V_{2...}$ V_N) over a global schema (G_S) as shown in Figure 10a. The mapping between the global schema and each local schema is expressed by associating the concepts in the local schemas as a view V_N over the global schema. For example, in Figure 10b, the global schema is defined as Global Disease Schema (G_S) (id, Name, serverityLevel, medName) which captures various diseases in terms of their unique identifier, commonly referred name, disease severity level, and medication name which acts as a treatment for this disease. The data for this global schema is obtained from the two local schemas and their respective views: a local schema $LS_{1,}$ Local Disease $Schema_1$ (LDS_1) (id, commonName, severityLevel), captures diseases in terms of identifier,

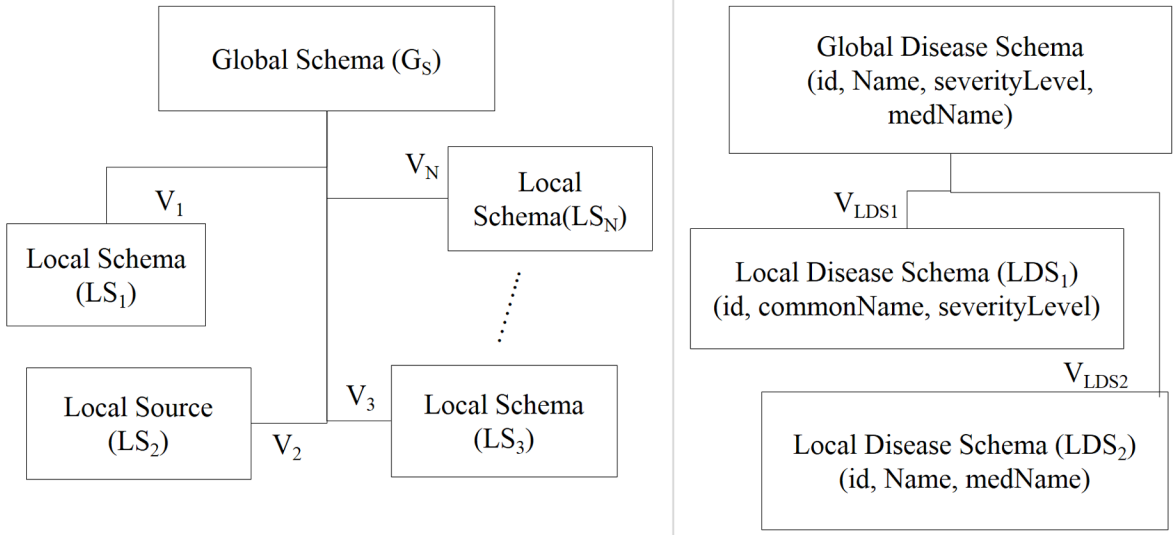
Figure 9. Facade software design pattern and its implementation



(a): *Façade Design Pattern.*

(b): *Façade implementation for complier.*

Figure 10. The LAV methodology and its implementation



(a): Local As View (LAV) approach for data integration.

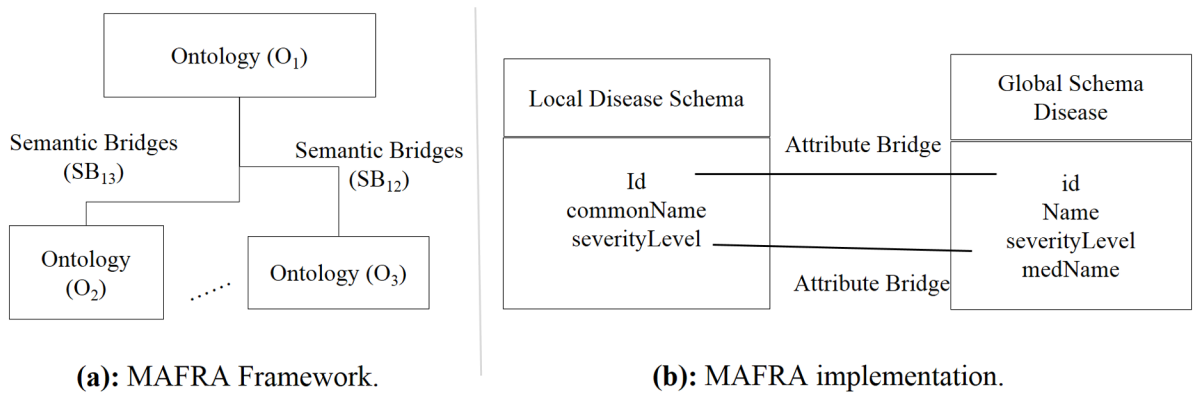
(b): LAV implementation.

commonly referred name, and disease severity level, respectively, and, its view V_{LDS1} is defined as a mapping (represented as ~) between LDS_1(commonName, severityLevel) ~ G_S(name, severityLevel). The second local schema LDS_2, Local Disease Schema$_2$ (LDS_2)(id, name, medName), captures the medications for respective diseases in terms of identifier, disease name, and medication name, respectively, and its view V_{LDS2} is defined as mapping between LDS_2(name, medName) ~ G_S(name, medName).

The MApping FRAmework (MAFRA) provides a conceptual framework for building semantic mappings between heterogeneous ontology models using *semantics bridges* as shown in the Figure 11a. A semantic bridge is a construct that allows the connections to have different meanings based on the needed interactions of the models. The mapping framework provides various types of semantic bridges such as RelationBridge, ConceptBridge, AttributeBridge, etc. For example, consider Local Disease Model (LDM) and the Global Disease Model (G_M) in Figure 11b. The semantically equivalent concepts between these models are mapped using attributeBridge i.e., LDM (id) ~ G_M(id), and as LDM(commonName) ~ G_M(Name) as shown in the Figure 11b; as a result, we are able to represent this higher level dependencies among different ontologies.

Pattern Description: The *Centralized Ontology Architectural Pattern (COAP)* as shown in Figure 12 consists of a Global Ontology Model (O_G), multiple local source ontology models (LO_1, LO_2, LO_3,……, LO_N), and the respective mappings (OM_1, OM_2,....,OM_N) between local ontology models and the global ontology (O_G). Conceptually, the COAP in Figure 12 can be aligned as: the Global Ontology Model (O_G) (similar to a Façade in Figure 9a), a global schema in Figure 10a, or an Ontology (O_1) in Figure 11a; the local source ontology models (LO_1, LO_2, LO_3,……, LO_N) can be local subsystems in Figure 9a, a local schemas from Figure 10a, or ontologies (O2,…,O_N) in Figure 11a; and, the mappings (OM_1, OM_2,....,OM_N) are similar to function calls

Figure 11. The MAFRA framework and its implementation



(a): MAFRA Framework. **(b):** MAFRA implementation.

between subsystems in Figure 9a, views (V_1,..,V_N) in Figure 10a, or semantic bridges in Figure 11a. The queries are generally performed on the Global Ontology Model (O_G) of COAP and the local sources are extracted using the mappings (OM_1, OM_2, …,OM_N) which are primarily *semantic* in nature, i.e., the mappings are mostly semantic queries similar to views in Figure 11b or semantic mappings in Figure 14. The global ontology model acts as a centralized reference model for mapping the local source ontology models which can also be traversed using the mapping to the global ontology model, thus eliminating the semantic interoperability issues between heterogeneous local ontology models. An enterprise application can use the Global Ontology Model supplemented by zero or more Local Ontology Models based on its needs.

Pattern Context and Usage: The COAP pattern is applicable where an enterprise application has an existing ontology acting as a global ontology and other ontology models (from potentially different enterprise applications) are to be integrated/mapped with the global ontology, or a single knowledge ontology model has to be built from existing multiple ontology models. An ideal example of the COAP is the Unified Medical Language System (UMLS) (Bodenreider, 2004) knowledge system developed and maintained by

Figure 12. Centralized Ontology Architectural Pattern

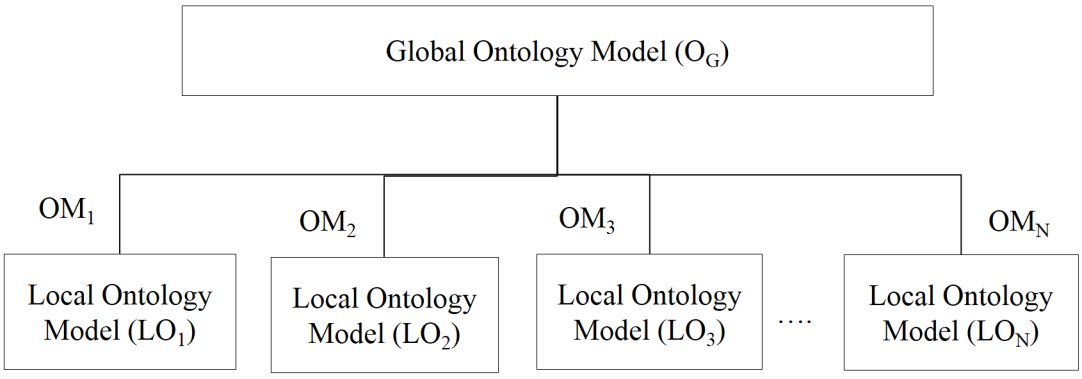
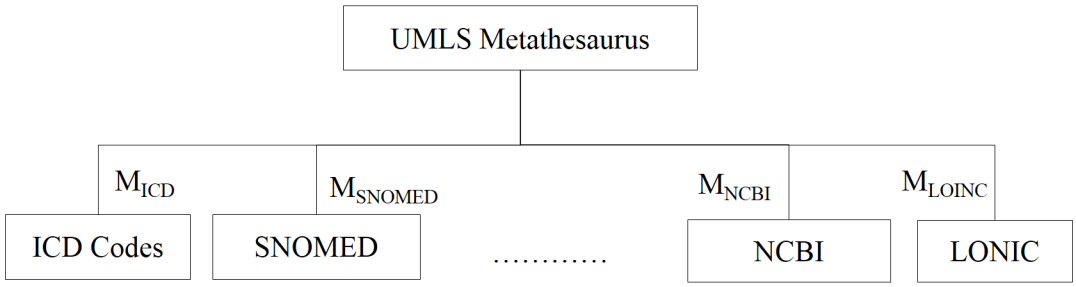


Figure 13. Unified Medical Language System as instance of COAP



National Institute of Health (NIH) which has two primary tools: UMLS Semantic Network (UMLS-SN) and UMLS Metathesaurus (UMLS-Meta). The UMLS-Meta holds the medical vocabulary obtained by aggregating existing medical standard ontologies such as ICD, DSM, OMIM, SNOMED-CT, etc., and also providing the mappings between these aggregated ontologies. As shown in Figure 13, the UMLS-Meta (UMLS Metathesaurus) can be viewed as a global ontology model (O_G, Figure 12), obtaining its medical vocabulary from various local source ontologies such as ICD Codes,, SNOMED-CT, NCBI, LOINC, etc., which are local ontology models (LO_1, LO_2,…,LO_N, Figure 12).

Using the mappings (OM_1, OM_2,…,OM_N) and the global ontology model (O_G) as the reference, the semantic mappings (implicit mappings) between the local source ontology models can be deduced. The semantic interoperability issues among the ontologies are eased by employing the defined mappings (OM_1, OM_2, …, OM_N) and the implicit mapping.

Another enterprise domain that could use COAP is e-commerce. While each e-commerce enterprise is unique from the perspective of its business model, sales strategies, software architecture, data management, user experience, etc., in total, a majority of them may overlap on the types of the merchandize/services offered (e.g., electronics, apparel, tools, books, materials, music, could space, streaming music, etc.). However, each enterprise system may structure the semantic knowledge about the merchandize/services offered differently, as it's primarily influenced by the interests of the enterprise and talent of the ontology developer. For example, an e-commerce service (ES_1, e.g., Amazon, Barnes & Noble, eBay, etc.) may have an ontology model(s) (O_{E1}) capturing the semantics knowledge about the merchandize/services offered by the enterprise system. Similarly, other e-commerce services (ES_2, ES_3, ES_4,……,ES_N) will have their own ontology models (O_{E1}, O_{E2}, O_{E3} .....,O_{EN}) capturing the semantic knowledge of their merchandize/services offered. When two enterprises, say ES_1 and ES_2, need to collaborate/merge, the semantic knowledge in the ontology models (O_{ES1} and O_{ES2}) has to be mapped to one another to ease the semantic interoperability issues (both Knowledge and Data). However, individual one-to-one mappings between different ESs are not a feasible solution, since it would require a custom bi-direction exchange in varied formats. For providing a more scalable and feasible solution, the enterprises need to collaborate to define a global ontology model (O_{EG}) and map their respective local ontology models (O_{E1}, O_{E2}, O_{E3} .....,O_{EN}) to the global ontology model. Thus, when multiple enterprises need to collaborate/merge, they can use the O_{EG} ontology model as a reference to ease semantic interoperability issues. The global ontology model O_{EG} will also act as a foundational platform for new enterprise systems and to interact with existing systems.

Pattern Application and Implementation: The NIH has provided open access to the UMLS system as previously discussed and shown in Figure 14 through a Web browser for online services and a Java Swing based UI supported with MySQL and Oracle database scripts for standalone applications as show in Figure 14. The sample database query diagrams between the UMLS-Meta schemas are shown in Figure 15, where the user can query: the *MRCONO* table for a common name for a given Concept Unique Identifier (CUI) for identifying medical concepts) as shown in Figure 15a; the *MRCONO* and *MRSTY* tables for finding all of the semantic types (UMLS-SN) for a given CUI as shown in Figure 15a; the MRCONO and MRDEF for obtaining the definition of a CUI as shown in Figure 15a; and, the MRCON and MRREL tables for obtaining all of the relationships a given concept is participating in using the Source Concept Unique Identifier (SCUI) that uniquely identifies the source of the medical concept as shown in Figure 15b. The UMLS acts as a global reference ontology model that can be utilized to obtain medical semantics and the system provides mappings between integrated local ontologies.

Figure 14. Unified Medical Language System as instance of COAP and its implementation

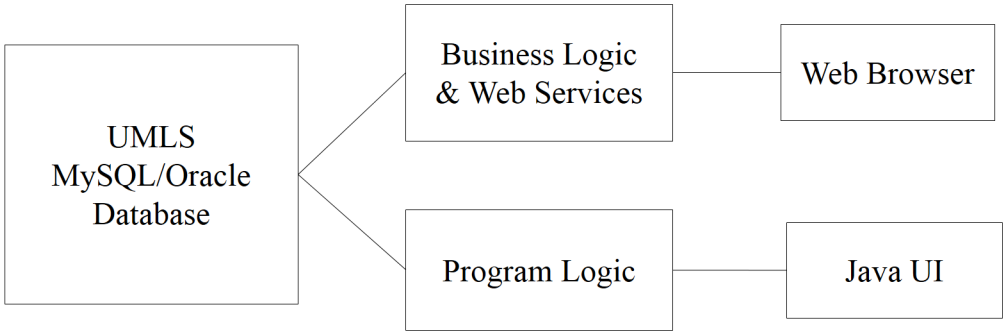
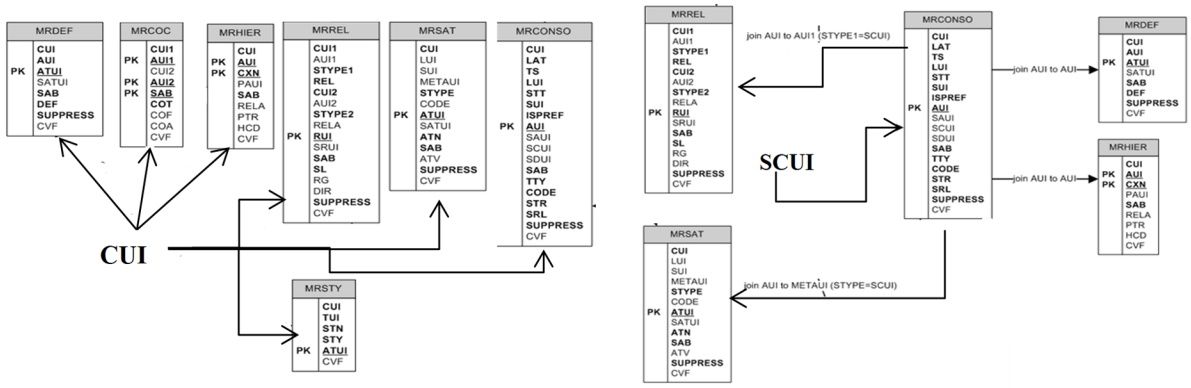


Figure 15. Sample database query diagrams for querying UMLS for a given CUI and SCUI



(a) Database query diagram for querying information for a given concept CUI.

(b) Database query diagram for querying information for a given source concept SCUI.

3.3. Layered Ontology Architectural Pattern

Pattern Name: Layered Ontology Architectural Pattern (LaOAP)

Pattern Motivation: The *Layered Ontology Architectural Pattern (LaOAP)* focuses on the underlying conceptual models for databases, software, and development in order to structure the ontology and its components in a layered manner. To motivate, we use a health care example in Figures 16a, 16b, and 16c, and an e-commerce/cloud example in Figures 16d, 16e, and 16f. First, if we start with databases, where an Entity Relationship Diagram (ERD) (Chen, 1976) is employed for modeling entities with respective attributes and associations limited by constraints (cardinality, primary key, etc.) on attributes and association in order to achieve the desired database behavior. As shown in Figure 16a, the entities Disease and Symptom are described using attributes id and name, and are connected using the has Symptom association with a many-to-many constraint. Similarly, in Figure 16d, the entities Customer (described with attributes cId and cEmail), CloudSpace (described using attributes location and space) and ContentAllowed (an enumeration) are connected with hasCloudSpace, and cloudAllows associations respectively with a many-to-many constraint imposed on them. The ERD models in Figure 16a and 16d can be modeled from an object-oriented perspective using UML Class Diagrams as shown in Figure 16b and 16e, respectively, where Disease, Symptom, Customer, CloudSpace, and ContentAllowed are all classes; id, cId, space (type Integer), and Name, Location (type String) are all attributes; and, they are all connected using the associations hasSymptom, hasCloudSpace, and cloudAllows. The OWL framework allows developers to add complex axioms such as disjoint, subset, union, not, etc., on the modeling elements themselves to further constrain and control behavior and data values. As shown in Figure 16c and 16f, Disease, Symptom, Customer, CloudSpace, and ContentAllowed are OWL Classes that are connected using the links hasSymptom, hasCloudSpace, and cloudAllows that are of type OWL objectProperty (equivalent to an association in UML), with an additional constraint stating that the pairs of classes Disease and Symptom and Customer and CloudSpace are disjoint. The OWL framework also allows experts to define Semantic Web Rules and Fuzzy Logic on the defined concepts. The ability of ERD, UML, and OWL to model from differing perspectives or layers can be employed to define an OAP that takes advantage of the layering of conceptual models, akin to ISO layers.

Pattern Description: The *Layered Ontology Architectural Pattern (LaOAP)* organizes various participating modules in the ontology in a layered fashion that separates the ontology from a functional perspective as shown in Figure 17a. The LaOAP pattern has five layers where the heart of the pattern is the *Conceptual Model Layer*. This layer holds the ontology conceptual model capturing the structure and semantics of the intended domain. The second layer is the *Axiom & Rule Layer* that captures the rules and constraints for semantically interpreting the ontology conceptual model entities. The third layer is the *Mapping Layer* that captures any semantic mapping between the current ontology model and any other target ontology models. The fourth layer is the *Terminology Layer* that contains the vocabulary for the ontology model captured in the Conceptual Model Layer and adhering to the rules of the Axiom & Rule Layer. Finally, the fifth layer

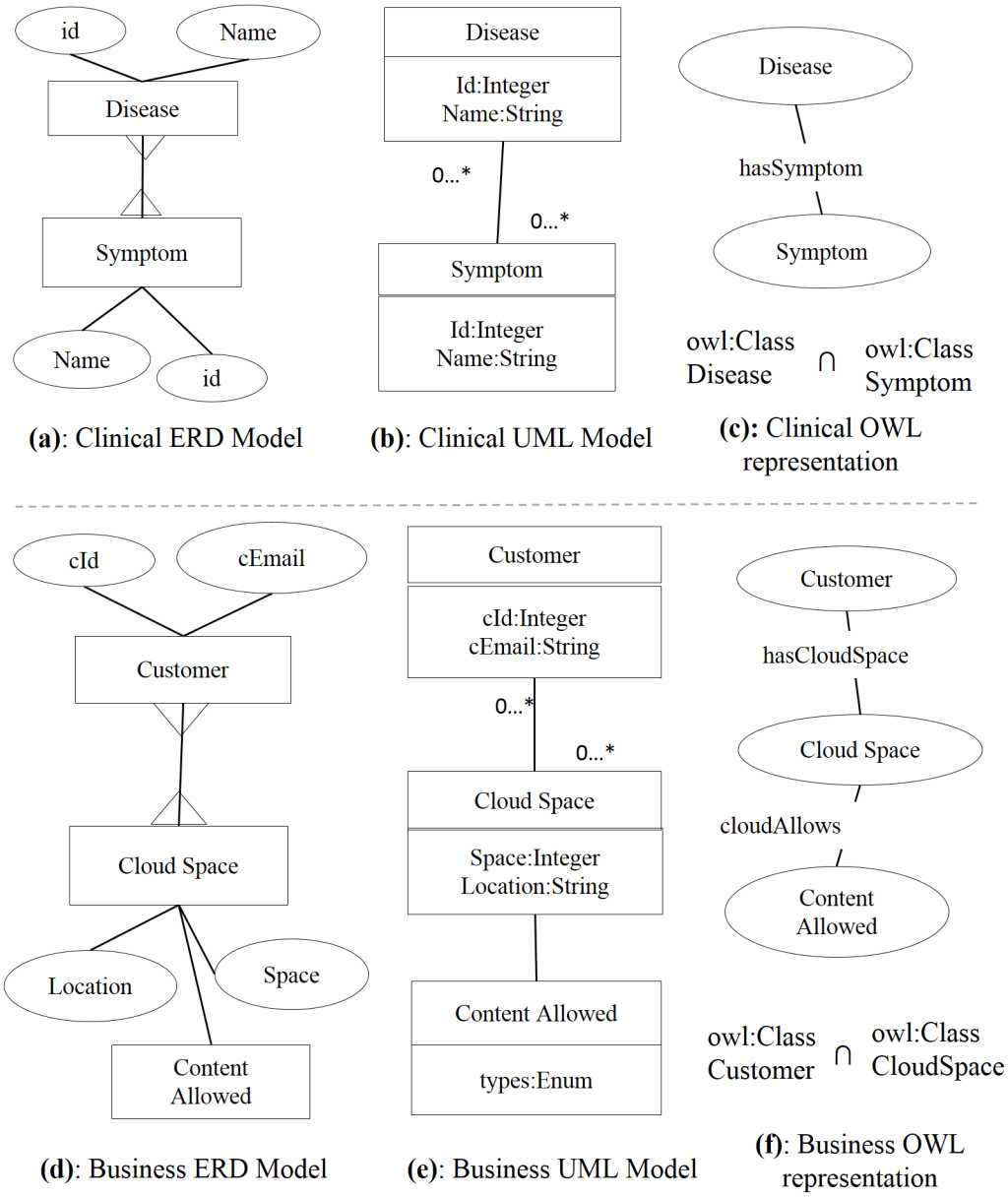
is the *Query and Web Service Layer* that allows users to query the conceptual model for its terminology or Web service API for serving Web-based requests.

For illustrating the five layers of the LaOAP pattern, consider the conceptual model for a health care application developed in OWL (Figure 16c and 16f) as shown in the Figure 17b. From the bottom up in Figure 17b, the *Conceptual Model Layer* holds the domain model (Disease Ontology Model) with classes (Disease, Symptom, etc.), associations (hasSymptom, hasMedication, etc.), and attributes (id, commonName, etc.). The *Axiom & Rule Layer* holds the first-order descriptive axioms for domain model entities such as the disjointness between Disease and Symptom classes. These axioms are defined in a new workspace by importing the ontology model from the *Conceptual Model Layer* to provide limited coupling between these two participating layers. The ontology designer can exploit this limited coupling to define multiple set of Axioms & Rules on the same ontology conceptual model. Continuing upward in Figure 17b, the *Mapping Layer* holds any mappings that are relevant and/or have been loaded to the current ontology model imported from *Conceptual Model Layer*. The attribute semantic bridge defined in the MAFRA framework (Figure 11) is utilized for illustrating an attribute equivalent mapping between the current Disease entity attribute id (Figure 16c) and the Disease entity attribute uid from a different conceptual Disease model. The *Terminology Layer* captures all of the instance data of the ontology model such as Heart Attack (instance of Disease class), Fever (instance of Symptom class), etc. Finally, at the top of Figure 17b, the *Query & Web Service Layer* has the domain model query logic (example OWL SPARQL queries, see Figure 9) for interrogating the Disease Ontology Model. In LaOAP, the outer layer may use the content defined in the inclusive inner layers. For example, the *Axioms and Rules Layer* will use the conceptual model defined in *Conceptual Model Layer*; the *Terminology Layer* and the *Query & Web service Layer* imports the *Conceptual Model Layer* and import *Axiom and Rules Layer*. The *Conceptual Model Layer* is required to define this pattern and all of the other layers are optional.

Pattern Context and Usage: The LaOAP can be used when an ontology developer requires minimal coupling between various participating ontological modules (Figure 17a), maximizing their reuse especially for the core ontology model in the *Conceptual Model Layer*. The intent is that the layers of functionality build upon one another from the Conceptual Model Layer to the Query & Web Service Layer. For instance, as shown in Figure 18, the UMLS-Meta schema can be reused by multiple health care institutions ($Hospital_1$, $Hospital_2$, …,$Hospital_N$) in order to develop customized sets of mapping and axioms specific to their enterprise application on top of the shared UMLS-Meta model. LaOAP allows maximum reuse of the core conceptual model of the ontology, thereby mitigating interoperability issues.

Pattern Application and Implementation: The implementation of LaOAP employing OWL (see Figures 16c and 16f again) to represent the ontology is shown in Figure 19. Starting from the bottom of Figure 19, the *Conceptual Model Layer* holds the domain model (classes Disease, Symptom, CloudSpace, etc.), associations (hasSymptom, hasCloudSpace, etc.), and attributes (id, commonName, spaceAllocated, etc.). The *Axiom & Rule Layer* holds the first-order descriptive axioms for model entities such as the disjointness between Disease and Symptom, the spaceAllocated attribute has an integer range, etc. These axioms are defined in a new workspace by importing the ontology model from the *Conceptual Model Layer* to provide limited coupling between these two

Figure 16. Domain models using ERD, UML and OWL



participating layers. The ontology designer can exploit this limited coupling to define multiple sets of Axioms & Rules on the same ontology conceptual model. For example, one business enterprise can define a range for the space attribute (CloudSpace class) between 2GB – 10GB, while another can define a different range between 5GB – 9GB. Note that both enterprises use the same ontology model, but can customize that from a constraint perspective.

Similarly, the enterprise can define various accepted values to ContentAllowed based on the business goals. The *Mapping Layer* holds any mappings that are relevant and/or have been

Figure 17. Layered Ontology Architectural Pattern and its implementation

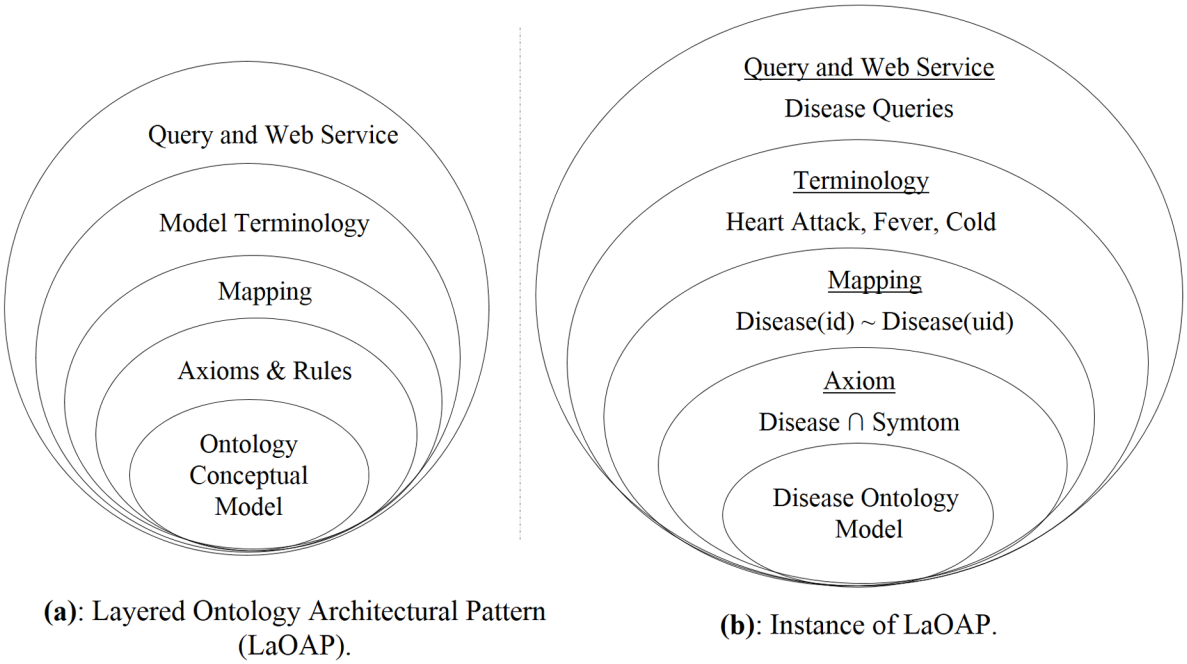
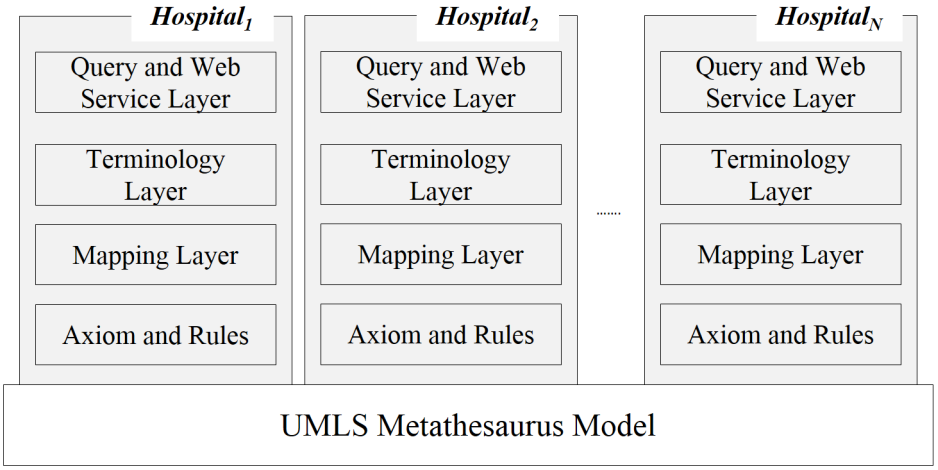


Figure 18. Illustrating LaOAP using UMLS-Metathesaurus (UMLS-Meta)



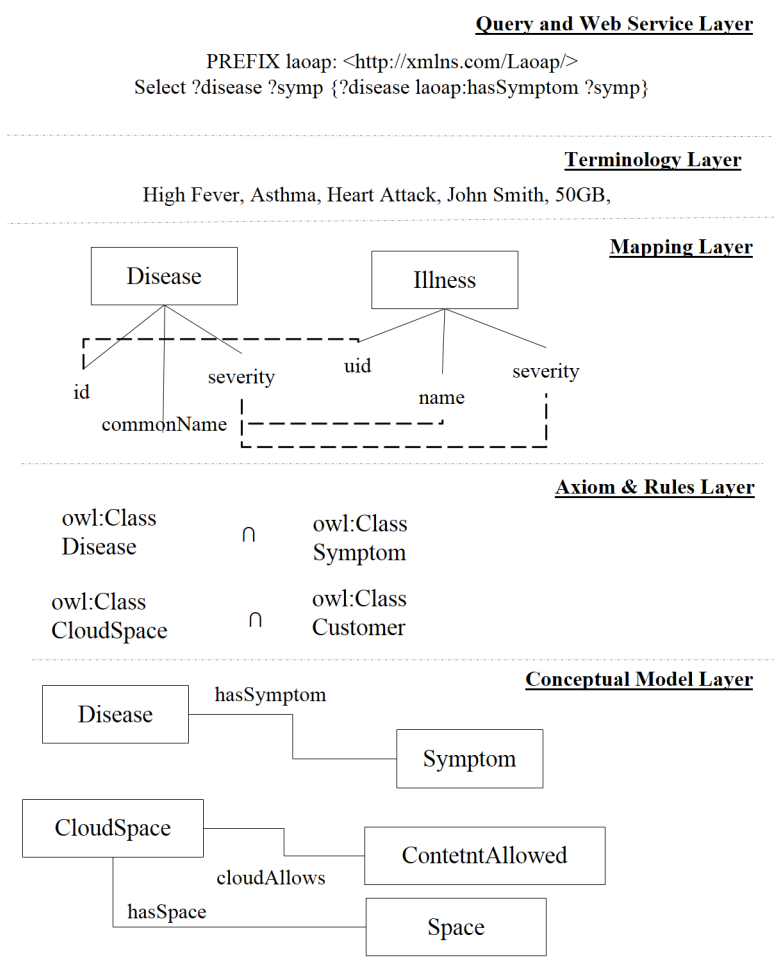
loaded to the current ontology model imported from *Conceptual Model Layer*. The attribute bridge defined in the MAFRA framework (Figure 10) is used for illustrating an attribute equivalent mapping between the current Disease entity (Figure 16c) and the Illness entity from a different conceptual model. The *Terminology Layer* captures all of the instance data of the ontology model such as Asthma (instance of Disease class), High Fever (instance of Symptom class), Heart Attack (instance of Disease class), 50GB (value of space attribute), etc. The ontology model and Axiom & Rules are imported into Terminology Layer to define the required domain vocabulary.

Finally, at the top of Figure 19, the *Query & Web Service Layer* can hold SPARQL query logic for interrogating the OWL conceptual model in the *Conceptual Model Layer*.

4. RELATED WORK

The work in this chapter has been influenced by a number of efforts and has ties to other work in ontology frameworks. First, in knowledge engineering, ontologies play a pivotal role in providing semantics to data, converting information into knowledge for a specific domain. To date, researchers have taken an approach to ontology patterns that implements different aspects of software design patterns (SDPs) in the domain of ontologies. The work of (Gangemi, 2005; Gangemi & Presutti, 2009) has defined Ontology Patterns (OP) and classified them into six categories: Structural OP, Correspondence OP, Content OP, Reasoning OP, Presentation OP, and Lexico-Syntactic OP. The Structural OP is equivalent to the structural SDP and further divided into Logical OP to handle the problem of expressivity and Architectural OP to affect the overall shape of the ontology either internally or externally. The Correspondence OP encompasses the Reengineering OP and provides a designer with a solution to the problem of transforming the ontology conceptual model. The

Figure 19. Implementation of LaOAP using OWL



Mapping OP refers to the possible semantic relations between mappable ontology model elements. Content OPs have gained priority as they solve knowledge design problems in terms of domain classes and properties, similar to Creational and Behavioral SDP. Our OAPs presented in this chapter can be positioned within the classification the six categories in (Gangemi, 2005) by placing the developed OAPs (LOAP, COAP and LaOAP) under Architectural OP, as they influence the architectural design of the ontology for an enterprise application. Our approach operates at a slightly higher level than their work which is more closely aligned to SDPs.

Second, the work on Ontology Patterns (Gangemi, 2005; Gangemi & Presutti, 2009) has proposed the Conceptual Ontology Design Pattern (CODeP) categorized as a Content OP to capture a generalized use case scenario acting as a template to solve domain knowledge design issues and offering a number of different patterns that target varied capabilities. For example, Time Indexed Participation is a CODeP that represents time indexing for the relation between persons and roles they play. The Role Task Pattern is also a CODeP representing temporary roles that objects can play, and the tasks that events/actions that are allowed to execute. The Participation Pattern is a CODeP extracted from the DOLCE ontology that illustrates participation relation between objects and events. The CODeP are knowledge-level abstractions that primarily influence the design of the ontology content and can be employed by OAPs to define ontology knowledge for the participating ontology models ($Ontology_1$, Figure 4; Global Ontology, Figure 12 and Layered Ontology, Figure 18). Similarly, both the paradigms CODeP and OAP provide abstractions at different levels, i.e., the former provides knowledge abstraction and the latter provide architectural abstraction.

Third, the work of (Clark, Thompson, & Porter, 2004) has proposed the concept of knowledge patterns defined as a semantic structure representing reoccurring patterns similar to SDP, but morphing the knowledge pattern entities onto domain classes instead of instantiating them. For example, a simple distribution knowledge patterns consists of a Producer P, a Switch mechanism S, and a Consumer C that are connected. This knowledge pattern is applicable to any domain model, e.g., P can represent a battery, generator, etc., S can be a common electrical switch, and C can be any electrical device such as light, heater, computer, etc. There is a close association among our LOAP and their knowledge pattern, since they are both similar to Pipes & Filters and Chain of Responsibility SDPs. However, their knowledge pattern primarily targets ontological concepts similar to CODeP for a domain across multiple enterprise applications that follows a similar knowledge structure.

Fourth, our work on a framework for ontologies has proposed extensions to the OWL framework to provide additional modeling features and introduce the concept of an ontology schema to evolve OWL to align to the UML meta-model (Saripalle, Demurjian, & Behre, 2011) with an associated software engineering process (Saripalle & Demurjian, 2012a) that elevates ontologies to be more design-oriented as opposed to instance based, allowing ontologies to be reused in an enterprise setting. As a foundation for the Ontology Architectural Patterns (OAP) in this chapter, our work (Saripalle & Demurjian, 2012) has introduced a Semantic Design Pattern defined as a modular domain knowledge pattern at the meta-model level which can be referenced while developing concrete domain models. The work in this chapter significantly extends the semantic design pattern concept to Ontology Architectural Patterns, promoting a design process for ontologies that has a much higher abstraction level than existing ontology frameworks.

Finally, there are a number of efforts underway in ontology frameworks related to Enterprise applications, where our work on OAP can be utilized to augment their approaches. The Enterprise Ontology (Uschold, 1995) project provides an

approach to model ontologies at the enterprise level, and is part of a larger Enterprise Project (Uschold, 1998) that is aimed at supporting a framework to collaboratively model enterprise applications. Our work in this chapter dovetails with this effort by providing the means to define OAPs within their framework to have a higher-level modeling (OAP) not currently supported by their work. Another effort, the Toronto Virtual Enterprise (TOVE) project (Gruninger & Fox, 1995) is an enterprise model that is able to generate common sense enterprise data models that has the ability to deduce answers to common sense queries regarding the enterprise. Our work on OAP in this chapter can augment TOVE and EO enterprise projects by promoting a higher-level abstract design process for ontologies (source of enterprise application knowledge for the projects) that supports knowledge reusability through ontology modeling coupled with architectural concepts. Additionally, the concept of OAP can easily be integrated into TOVE's project lifecycle model to tackle any interoperability issues. For the field of Ontology Based Information Extraction (OBIE) (Maynard, Yankova, Kourakis, & Kokossis, 2005; Saggion, Funk, Maynard, & Bontcheva, 2007; Wimalasuriya & Dou, 2010), the construction of an application for an enterprise domain heavily relies on ontology models that capture the required domain knowledge using information and knowledge extracted from unstructured and semi-structured sources (e.g., Web, texts, data streams, etc.). Our work on OAP can provide OBIE with the ability to model knowledge at a higher level of abstraction, providing another means to organize the needed knowledge into an ontology via a pattern. Similarly, in enterprise data warehousing architecture (Blechner, Saripalle, & Demurjian, 2012), ontologies play a key role of attaching semantics to the stored data for: providing meaning to information, support querying of the stored data using the shared standard concept semantics, and providing the ability to present the stored information as human readable knowledge through semantic definition. We envision that our OAP architectural pattern framework will assist ontology designers to make better ontology architectural decisions in the initial phases of the ontology life cycle process that will influence the usage of the ontology models in multiple enterprise application setting, thus primarily targeting enterprise interoperability.

5. FUTURE RESEARCH DIRECTIONS

Over the span of the last 30 years, the computing field has evolved to be more domain independent and device agnostic trending towards the development of software, database, and Web applications that are easily modified and evolved as requirements change over time. There have been many examples in computing where industry and academia have come together towards generalized solutions. The 1980s provided a transition from procedural to object-oriented languages (objective-C, Eiffel, GNU C++, AT&T C++, etc.). Standards eventually coalesced C++ into a single solution. During the same time, there was a wide (and confusing) collection of object-oriented design models with different (and often conflicting) concepts and terminology that were eventually unified into UML. The Object Management Group (OMG) took control of the standard, which included a UML meta-model and the ability to generate an XML instance of a UML design with both the content of the design and the positioning of the graphical objects in each UML diagram that was easily ported. There was a similar convergence in the database community, first on standardizing the Structure Query Language (SQL) and later in the use of XML and XMI to export/import both a database schema and the tuples themselves from one database system to another. In the late 1990s, Java came on the scene to revolutionize programming (write once, run anywhere), and today its dominance throughout computing and our daily lives is significant particular in regards to mobile

devices. This history serves as a strong justification as to why there needs to be improvements in the ontology development process towards a design-oriented approach where reuse can be facilitated and interoperability can be achieved.

In order to ease structural and semantic interoperability issues among knowledge sources, we need to analyze and implement the lessons learned from software engineering, programming languages, and computing. *First*, there is definitely a lack of *commitment* and *agreement* by knowledge developers and domain experts towards a common knowledge source. This is evident in the health care domain as shown in Figure 1 and for many other domains such as e-commerce, digital collections (METS), human resource applications (HR-UML), legal applications (LegalUML), document exchange (Open Office XML), where multiple standards are developed for providing semantic knowledge on the same domain with minimal knowledge reuse. We need to strive towards a *formal standard* for knowledge sources, primarily ontologies, as they are currently the key source of knowledge semantics. This process encompasses the development of standard knowledge models and, most importantly, employing up-to-date technologies such as UML 2.0, RDF, OWL, etc. For example, many existing medical standards are implemented using outdated technologies such as XML DTDs, KIF, Frames, Semantic Networks, etc.; the future will require knowledge representation frameworks to demonstrate the same evolution as UML tools and databases towards a common context.

Second, even with the existence of multiple standard models for a given domain, there would still be disagreement on the semantics on the elements defined in the standard. For example, the health care domain has various representational standards such as CCD, CCR, HL7 CDA for patient data and ontologies with similar domain interests such as SNOWMED-CT, LOINC, ICD, DSM, etc. However, these standards lack the agreement on semantics of the defined concepts; CCD and CCR store the same information in different ways and as a result infer different semantic relationships among the information that comprises a patient's medical record. Further, it is nearly impossible to take data form one vendor's EHR to another vendor's EHR since data formats and ontologies are all proprietary. We need to understand the existing standards from both structural and semantic perspectives, and define an agreed standard *meta-model* for enterprise domains with sound structure and semantics that can be utilized in a unification process. The structure and semantics for this meta-model can be defined based on existing domain model standards supplemented with additional concepts. This developed meta-model will then act as a standard framework for developing multiple standard models. For example, the UML meta-model provides a UML Profile feature which is a generic lightweight extension mechanism allowing developers to tailor the UML metamodel to domain specific requirements. The extension allows refining the standard UML semantics according to user requirements in a strictly additive manner without contradicting the defined metamodel semantics. The OMG employs this feature to define profiles (Fuentes-Fernández & Vallecillo-Moreno, 2004) in various domains such as Service oriented architecture Modeling Language (SoAML), Distributed Data Systems (DDS), Advanced and Integrated Telecommunication Services (TelcoML), etc. These UML Profiles based on the UML metamodel act as a domain specific standard metamodel with well-defined concepts from respective domains for developing multiple domain models. Thus, developing a generic or domain specific metamodel for domains such as e-commerce, human resources, legal field, medicine, finance, business, biology etc., will ease interoperability issues as the *structure* and *semantics* defined in the metamodel are well agreed among the domain experts.

Third, we need to encourage domain experts to develop and standardize *knowledge patterns* which can be referred to while developing knowl-

edge models for specific domains. When multiple knowledge models refer to the same knowledge pattern, the semantics of the defined model can easily be understood, closing the gap for any semantic misinterpretations. As a result, knowledge patterns supplement ontology patterns and strive towards a process that emphasizes design rather than development. For example, the well-defined Task-Role CODeP proposed by Gangemi in Figure 20 in Section 4 can be referenced for developing any domain applications involving tasks (such as financial transactions, prescription writing, changes to Web services, etc.), roles (such as manager, physician, developer, etc.), object (such as currency, patient, software code), etc. Similarly, the domain experts need to develop, document, maintain, and share domain knowledge patterns for easing semantic interoperation.

CONCLUSION

This chapter has addressed a serious disconnect between ontologies for enterprise applications that are primarily focused on encoding the captured domain knowledge concepts and their respective relationships at the instance level and software-engineering-based approaches focused on developing domain models which provide abstract view of the solution employing software design patterns. The current approach chosen by ontologists focuses on the development of specific instance-level ontologies to capture knowledge requirements of specific domain application but with the side effect of causing potential *structural* and *semantic interoperability* conflicts when one attempts to integrate two or more ontologies from different contexts. For two or more enterprise application that need to share data and knowledge via their respective ontologies, this is a significant roadblock to facilitate interoperation. There is a clear need to support Enterprise Interoperability (EI) particularly in regards to the interoperation of data and knowledge across multiple enterprise applications. The major premise of this chapter is that there needs to be an upgrade from an ontology development process to one that is focused on abstraction and the leveraging of design models (UML and ER) and approaches (SDPs). Through this enhanced design and modeling process, ontologies will be able to be more clearly and abstractly defined with the potential for reuse in multiple settings. The work of this chapter applied and extended the concept of SDPs in order to propose a set of Ontology Architectural Patterns (OAPs) that would provide ontologists with a significant abstraction capability to be able to more effectively design ontologies that can promote enterprise interoperability in terms of both data and knowledge.

Towards this objective, Section 2 presented background and motivation on the different ways that ontologies can be exploited in the design and development process, reviewed software design patterns, and provided a context for the work of this chapter by exploring enterprise interoperability (EI) with a particular emphasis on data and knowledge integration via ontologies. Using this as basis, in Section 3, we presented three Ontology Architectural Patterns (OAPs): the Linear Ontology Architectural Pattern (LOAP) in Section 3.1 for modeling the linear/parallel architectural arrangement of ontology models for achieving the required knowledge goal; the Centralized Ontology Architectural Pattern (COAP) in Section 3.2 which supported the definition of a centralized ontology model and its interactions with multiple local/other ontology models; and, the Layered Ontology Architectural Pattern (LaOAP) in Section 3.3 that defined a layered architectural arrangement of participating modules in the ontology model. The discussed OAPs applied concepts from structural and architectural software design patterns in order to define the three OAPs that allow ontologies to be both modeled and related to one another at a higher conceptual level. As a result, OAPs promote the design of domain knowledge semantics into a modular concept that can then

yield reusable ontology models for enterprise applications to more effectively address *structural* and *semantic* enterprise interoperability. To place our work into its proper perspective, Section 4 explored related work in ontology patterns and ontology frameworks, comparing their efforts to our OAPs thereby demonstrating the inclusion of our OAPs in practice. Section 5 focused on future directions by first providing a discussion of the history of programming languages, software models, and databases, and that the standards have shaped their evolution over time. This in turn serves as a basis for discussing improvements in regards to ontology design and development to work towards a knowledge representation framework that operates in a more abstract perspective through the proposal of OAPs. While the chapter demonstrated the developed OAP in the health care domain, the work is more general in nature and applicable to interoperability issues among ontologies for, any enterprise domain. This argument was supported by providing additional and complementary examples of LOAP, COAP, and LaOAP for e-government, e-services, and e-commerce domains. If we can continue to move the ontology process to embrace a more conceptual and design level perspective, there is great potential to allow ontologies to be more easily integrated leading to an ability to share information across a domain.

REFERENCES

Allemang, D., & Hendler, J. (2011). *Semantic web for the working ontologist: Effective modeling in RDFS and OWL* (2nd ed.). Waltham, MA: Morgan Kaufmann.

Ashburne, M., & Lewis, S. (2002). On ontologies for biologists: The gene ontology-untangling the web. In *Proceedings of the Novartis Found Symposium* (pp. 66-80). Novartis Found.

ASTM. (2003). *Standard specification for continuity of care record (CCR)*. Retrieved from www.astm.org/Standards/E2369.htm

Baader, F., McGuinness, D., Nardi, D., & Patel-Schneider, P. (2007). *The description logic handbook: Theory, implementation and applications*. New York, NY: Cambridge University Press. doi:10.1017/CBO9780511711787

Bell, M. (2008). *Service-oriented modeling: Service analysis, design, and architecture*. Hoboken, NJ: Wiley & Sons. doi:10.1109/EDOC.2008.51

Bettahar, F., Moulin, C., & Barthes, J. (2009). Towards a semantic interoperability in an e-government applications. *Electronic. Journal of E-Government*, *7*(3), 209–226.

Blechner, M., Saripalle, R., & Demurjian, S. (2012). Proposed star schema and extraction process to enhance the collection of contextual & semantic information for clinical research data warehouses. In *Proceedings of the 2012 International Workshop on Biomedical and Health Informatics.* Philadelphia: Academic Press.

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

Brauer, M., & Schubert, S. (2013). *The OpenOffice.org XML project*. Retrieved from http://www.openoffice.org/xml/

Burstein, H. M., & McDermott, V. D. (2005). Ontology translation for interoperability among semantic web services. *AI Magazine*, *26*(1), 71–82.

Charalabidis, Y., Goncalves, R. J., & Popplewell, K. (2010). Developing a science base for enterprise interoperability. In K. Popplewell, J. Harding, C. Ricardo, & R. Poler (Eds.), *Enterprise interoperability IV: Making the internet of the future for the future of enterprise* (pp. 245–254). Springer Publications. doi:10.1007/978-1-84996-257-5_23

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

Chiang, M. F., Hwang, J. C., Yu, A. C., Casper, D. S., Cimino, J. J., & Starren, J. (2006). Reliability of SNOMED-CT coding by three physicians using two terminology browsers. In *Proceedings of the 2006 AMIA Annual Symposium* (pp. 131-135). AMIA.

Clark, P., Thompson, J., & Porter, B. (2004). Knowledge patterns. In S. Staab, & R. Struder (Eds.), *Handbook on ontologies* (pp. 191–207). Berlin: Springer. doi:10.1007/978-3-540-24750-0_10

Davis, J., Harris, S., Crichton, C., Shukla, A., & Gibbons, J. (2008). Metadata standards for semantic interoperability in electronic government. In *Proceedings of the 2nd International Conference on Theory and Practice of Electronic Governance* (pp. 67-75). Academic Press.

Demurjian, S., Saripalle, R., & Behre, S. (2009). An integrated ontology framework for health information exchange. In *Proceedings of the 21st International Conference on Software Engineering and Knowledge Engineering* (pp. 575-580). Boston: Academic Press.

DSM. (2012). *DSM-5 implementation and support*. Retrieved from http://www.dsm5.org/

FIBO. (2013). *Financial report ontology*. Retrived from http://financialreportontology.wikispaces.com/home

Fonou-Dombeu, J. V., & Huisman, M. (2011). Semantic-driven e-government: Application of uschold and king ontology building methodology for semantic ontology models developments. *International Journal of Web & Semantic Technology*, *2*(4), 1–20. doi:10.5121/ijwest.2011.2401

Freeman, E., Robson, E., Bates, B., & Sierra, K. (2004). *Head first design patterns*. Sebastopol, CA: O'Reilly Media.

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

Gali, A., Chen, C. X., Claypool, K. T., & Uceda-Sosa, R. (2004). From ontology to relational databases. In S. Wang (Ed.), *ER workshops (LNCS)* (Vol. 3289, pp. 278–289). Berlin: Springer-Verlag.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: Elements of reusable object-oriented software*. Boston, MA: Addison-Wesley.

Gangemi, A. (2005). Ontology patterns for semantic web content. In *Proceedings of the 4th International Semantic Web Conference* (pp. 262-276). Academic Press.

Gangemi, A., & Presutti, V. (2009). Ontology design patterns. In S. Staab, & R. Struder (Eds.), *Handbook on ontologies: International handbooks on information systems* (pp. 221–243). IOS Press. doi:10.1007/978-3-540-92673-3_10

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.

HL7 CDA R2. (2008). *HL7/ASTM implementation guide for CDA® R2 -Continuity of care document (CCD®) release 1*. Retrieved from http://www.hl7.org/implement/standards/product_brief.cfm?product_id=6

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

HR-XML. (2013). *HR-XML consortium*. Retrieved from http://www.hr-xml.org/

Hu, B. (2010). Semantic interoperability for financial information: A component-based approach. In *Proceedings of 3rd IEEE International Conference on Computer Science and Information Technology* (pp. 228- 232). IEEE.

ICD. (2013). *ICD-10*. Retrieved from http://www.cms.gov/Medicare/Coding/ICD10/index.html?redirect=/icd10

Janowicz, K., Scheider, S., Pehle, T., & Hart, G. (2012). Geospatial semantics and linked spatio-temporal data – Past, present, and future. *Semantic Web – Interoperability, Usability. Applicability*, *3*(4), 321–332.

Jardim-Goncalves, R., Grilo, A., Agostinho, C., Lampathaki, F., & Charalabidis, Y. (2013). Systematisation of interoperability body of knowledge: The foundation for EI as a science. *Special Information Systems for Enterprise Integration, Interoperability and Networking. Theory and Applications*, *7*(1), 7–32.

Java Ontology Editor. (1998, May 19). *Java ontology editor (JOE)*. Retrieved from http://cit.cse.sc.edu/demos/java/joe/joeBeta-jar.html

Kohler, M. (2007). *UMLS for information extraction.* (Mater's Thesis). Vienna University of Technology, Vienna, Austria.

Krauthammer, M. (2002). *Brief review of clinical vocabularies*. Retrieved from http://www.cbil.upenn.edu/Ontology/MKreview.html

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*. Victoria, Canada: Trafford Publishing.

Legal, X. M. L. (2008). *Overview of the OASIS LegalXML*. Retrieved from http://www.legalxml.org/

Lenzerini, M. (2002). Data integration: A theoretical perspective. In *Proceedings of the 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (pp. 233-246). ACM.

LOC. (2012). *Standards in the library of congress*. Retrieved from http://www.loc.gov/standards/

LOINC. (2013). *Logical observation identifiers names and codes (LOINC®)*. Retrieved from http://loinc.org/

Maedche, A., Motik, B., Silva, N., & Volz, R. (2002). MAFRA - A mapping framework for distributed ontologies. In *Knowledge engineering and knowledge management: Ontologies and the semantic web (LNCS)* (Vol. 2473, pp. 235–250). Berlin: Springer. doi:10.1007/3-540-45810-7_23

Makela, T., Rommel, K., Uskonem, J., & Wan, T. (2007). *Towards a financial ontology – A comparison of e- business process standards.* Retrived from http://www.soberit.hut.fi/T-86/T-86.5161/2007/FinancialOntology_final.pdf

Maynard, D., Yankova, M., Kourakis, R., & Kokossis, A. (2005). Ontology-based information extraction for market monitoring and technology watch. In *Proceedings of the Workshop Of ESWC, End User Apects of Semantic Web*. Heraklion.

Nagarajan, M., Verma, K., Sheth, A., Miller, J., & Lathem, J. (2006). Semantic interoperability of web services – Challenges and experiences. In *Proceedings of the 4th IEEE International Conference on Web Services* (pp. 373-382). IEEE.

OWL-S. (2004, November). *OWL-S: Semantic markup for web services*. Retrieved from http://www.w3.org/Submission/OWL-S/

Popplewell, K., Lampathaki, F., Koussouris, S., Mouzakitis, S., Charalabidis, Y., Goncalves, R., & Agostinho, C. (2012). *ENSEMBLE: Promoting future internet enterprise systems research.* Retrieved from http://www.fines-cluster.eu/fines/jm/Publications/Download-document/339-ENSEMBLE_D2.1_EISB_State_of_Play_Report-v2.00.html

Powers, S. (2003). Practical RDF. Sebastopol, CA: O' Reilly Media.

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

Saggion, H., Funk, A., Maynard, D., & Bontcheva, K. (2007). Ontology-based information extraction for business intelligence. In *Proceedings of the 6th International Conference on Semantic Web.* IEEE.

Saripalle, R., & Demurjian, S. (2012a). Towards a hybrid ontology design and development life cycle. In *Proceedings of the 11th International Conference on Semantic Web and Web Services.* IEEE.

Saripalle, R., & Demurjian, S. (2012b). Semantic patterns using the OWL domain profile. In *Proceedings of the 2012 International Knowledge Engineering Conference* (pp. 3-9). IEEE.

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

Smith, B. A., & Ceusters, W. B. (2006). Ontology as the core discipline of biomedical informatics: Legacies of the past and recommendations for the future direction of research. In G. D. Crnkovic, & S. Stuart (Eds.), *Computing, philosophy, and cognitive science*. Cambridge, UK: Cambridge Scholars Press.

SNOMED CT. (2013). *SNOMED CT® technical implementation guide*. Retrieved from http://ihtsdo.org/fileadmin/user_upload/doc/

SOAP. (2007, April). *Simple object access protocol*. Retrieved from http://www.w3.org/TR/soap/

Wimalasuriya, D. C., & Dou, D. (2010). Ontology-based information extraction: An introduction and a survey of current approaches. *Journal of Information Science*, *36*(6), 306–323. doi:10.1177/0165551509360123

WSDL. (2001, March). *Web services description language*. Retrieved from http://www.w3.org/TR/2001/NOTE-wsdl-20010315

ADDITIONAL READING

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

Bendaoud, R., Napoli, A., & Toussaint, Y. (2005). A proposal for an Interactive Ontology Design Process based on Formal Concept Analysis. In *Proceedings of the Formal Information and Ontology System (FIOS).*

Boone, K. (2011). *The CDA book*. New York, NY: Springer. doi:10.1007/978-0-85729-336-7

D'Amore, J. D., Sittig, D. F., Wright, A., Iyengar, M. S., & Ness, R. (2011). The Promise of the CCD: Challenges and Opportunity for Quality Improvement and Population Health. In *Proceedings of the 2011 AMIA Annual Symposium* (pp. 285-294).

Davis, J., Harris, S., Crichton, C., Shukla, A., & Gibbons, J. (2008). Metadata standards for semantic interoperability in electronic government. In *Proceedings of the 2nd international conference on Theory and practice of electronic governance* (pp. 67-75).

Djurić, D., Gašević, D., & Devedžić, V. (2005). Ontology Modeling and MDA. *Journal of Object Technology*, *4*(1), 109–128. doi:10.5381/jot.2005.4.1.a3

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

Gruber, R. T. (2005). Toward Principles for design of ontologies used for knowledge sharing. *Journal of Human Computer Studies*, *43*, 90–928.

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

Hartmann, J., Palma, R., & Sure, Y. (2005). OMV–Ontology Metadata Vocabulary for the Semantic Web. In *Proceedings of the International Workshop on Ontology Patterns for the Semantic Web.*

Herre, H., Heller, B., Burek, P., Hoehndorf, R., Loebe, F., & Michalek, H. (2007). *General Formal Ontology (GFO), A Foundational Ontology Integrating Objects and Processes. Part I: Basic Principles. Research Group Ontologies in Medicine (Onto-Med)*. University of Leipzig.

Mazzoleni, P., Crispo, B., Sivasubramanian, S., & Bertino, E. (2008). Xacml policy integration algorithms. [TISSEC]. *ACM Transactions on Information and System Security*, 11.

Nicola, A., Missikoff, M., & Navigli, R. (2005). A Proposal for a Unified Process for Ontology building: UPON. In *Proceedings of the 16th International Conference on Database and Expert Systems Applications.*

OMIM. (2013). Retrieved from Online Mendelian Inheritance in Man: http://www.omim.org/

Trinh, Q. (2007). Semantic Interoperability Between Relational Database Systems. In *Proceedings of the 11th International Database Engineering and Applications Symposium* (pp. 208-215).

Wache, H., Vögele, T. U., Stuckenschmidt, H., Schuster, G., Neumann, H., & Hübner, S. (2001). Ontology-Based Integration of Information - A Survey of Existing Approaches. In *Proceedings of the International Workshop on Ontologies and Information Sharing* (pp. 108-117).

KEY TERMS AND DEFINITIONS

Continuity of Care Record (CCR): A document standard for health information typically used for representing data in Personal Health Records (PHR).

Electronic Health Record (EHR): An electronic health record contains all related health information, from medications to procedures, and is managed by the institution in which it is stored (e.g. hospital, private practice, clinic, etc.).

Enterprise Interoperability (EI): A field of activity with the aim to improve the manner in which enterprises, by means of information and communications technologies, interoperate with other enterprises, organizations, or with other business units, in order to conduct their business.

eXtensible Markup Language (XML): A structured language utilized for information exchange, standards and information validation via the use of schemas. Its extensibility allows developers and experts to design and implement common standards for the use across systems and domains.

Health Information Exchange (HIE): The ability to share information among health information technology systems by linking information for the same patient across multiple repositories to provide a complete health care view.

Health Language Seven Clinical Document Architecture (HL7 CDA): HL7 CDA is an XML-based markup standard intended to specify the encoding, structure and semantics of clinical documents for exchange.

Interoperability: The ability of diverse systems and organizations to work in a collaborative environment.

Resource Description Framework (RDF): RDF is as a metadata data model which is used for conceptual description or modeling of information that is implemented in web resources, using a variety of syntax notations and data serialization formats.

Software Design Patterns (SDPs): SDPs are general reusable solution to a reoccurring problem in multiple different situations with similar context without involving any application specific objects.

SPARQL Protocol and RDF Query Language (SPARQL): SPARQL is an RDF query language that will be able to retrieve and manipulate data stored in RDF/OWL format. SPARQL allows for a query to consist of triple patterns, conjunctions, disjunctions, and optional patterns.

Systematized Nomenclature of Medicine Clinical Terms (SNOMED-CT): an organized computer processable collection of medical terms providing codes, terms, synonyms and definitions covering domains such as diseases, findings, procedures, microorganisms, substances, etc.

Unified Medical Language System (UMLS): UMLS is defined as a compendium of multiple standard medical vocabularies such as ICD, LOINC, SNOMED-CT, etc., and provides a mapping structure between the integrated standards.

Web Ontology Language (OWL): The Web Ontology Language is a knowledge representation languages for defining ontologies.