CHAPTER 2

Literature Review

2.1 Introduction

Ontological engineering has garnered increasing attention over the last few years, as researchers have recognized ontologies. Ontologies are not just for natural language processing, metaphysics, common sense knowledge and enterprise modelling etc. [DEV02, FCM+03]. However, it caters software engineers in modelling various applications of the world and hence can make use of ontologies to obtain knowledge based applications [CJB99, FGD92, FMR98]. In addition, a recent survey of the field suggests developers of practical software systems may especially benefit from ontology use [FH97, MA05, NHM00]. This survey earmarked several application classes of software engineering that benefit from using ontologies. These classes include generalized information system modelling, software requirement engineering, software reuse, software reliability, software security and abstraction [ABH+99, BOE96, DEV02, FMR98].

Ontologies are explicit representations of domain concepts and provide basic structure or armature around which knowledge based system can be constructed [ST99]. Ontology is a system of concepts and relations, in which all concepts are defined and interpreted in a declarative way [DEV02, GFC04]. System defines the vocabulary of a problem domain and a set of constraints that can be combined to model a domain. In a distributed environment, agents use ontologies to establish communication at the knowledge level using specific languages and protocols [BTD04]. Ontologies are explicit representations of agents’ commitments to a model of the relevant world and hence enable knowledge sharing and reuse [JMY99, NFF+91, LINK9].
Ontological engineering encompasses a set of activities conducted during conceptualization, design, implementation and deployment of ontologies. Ontological engineering covers topics including knowledge representation formalisms, development methodology, knowledge sharing and reuse, knowledge management, business process modelling, systematization of domain knowledge, information retrieval, standardization, and evaluation [MIZ98, NFF+91, HOO98]. It also provides design rationale of a knowledge base, helps to define the essential concepts of the world of interest, allows for a more disciplined design of a knowledge based system, and enables knowledge accumulation [DW99, GOM01, SSS+01].

Several special issues of journals and magazines dedicated to the field of ontologies have described current trends in the field of ontologies. These include creating large-scale ontologies, defining expressive languages for representing ontological knowledge, and implementing systems that support ontology-based applications [CJB98, CJB99, LGS+99, LEN95, SRK+97, VRM+99]. Unfortunately, a vast majority of these articles have not covered the relations between ontological engineering and other software engineering areas. As a result, specialists from other disciplines struggle to understand the benefits of ontologies and to map the terminology of ontological engineering to their own fields.

The organization of this chapter is done as follows: Section 2.2 elaborates ontological engineering, its themes and aspects of two general disciplines that help to develop ontologies at specification and conceptualization stage such as modelling and metamodelling. Section 2.3 emphasizes on goals of software practitioners. Moreover, it attempts to clarify the skills useful for practicing ontology based software engineering. Various disciplines of software engineering which cater by ontology are discussed in Section 2.4. Finally, we conclude in Section 2.5 with the summary.
2.2 Ontological Engineering

Ontological engineering field has been subject to considerable study and research during the last decade. It is observed that, ontological engineering refers to the set of activities that concern the ontology design principles such as clarity, minimal encoding bias, extendibility, and coherence [UG96, DEV02, GFC04]. According to principle of clarity, ontology should communicate effectively the intended meanings of defined terms. Also, definitions should be objective and defined by necessary and sufficient conditions. Whereas, minimal encoding bias principle suggests that the conceptualization should be done at knowledge level without depending on a particular symbol level encoding. Then, in line with principle of extendibility, ontology designer should be able to define new terms for special uses based on existing vocabulary. Lastly, principle of coherence states that ontology should sanction inferences that are consistent with the existing definitions [GRU92]. Now, we describe various themes of ontological engineering in this section.

2.2.1 Themes

Ontological literature on ontologies and ontological engineering usually covers the concepts shown in Figure 2.1. While, we put ontological engineering in the context of other disciplines, many similarities and analogies arise. These similarities allow practitioners to make connections between ontological engineering and other disciplines, to bridge comprehension gaps, and to see known concepts and practices in another light.

Naive Physics and Commonsense Knowledge

Ontology about naive physics is defined as the ontology for liquids developed by Hayes [HAY85]. Naive physics can be hard to formalize such as the problem with knowledge about liquids is that they have no definite shape and can merge split mix in mysterious ways. Formalizations of knowledge about physical objects can be found in [CLA81, S87, CRC95, BBW96b]. Ontologies of
microscopic and macroscopic views on the electrical domain are combined by Liu [LIU92]. Conversely, aim of Cyc project is to build up a large knowledge base with commonsense knowledge [LG90, LINK9]. In addition, to help structuring knowledge in the knowledge base, ontology of common-sense top-level concepts have been developed [HOB85, DAV90].

**Engineering and Technical Applications**

Ontologies have been developed for engineering and technical applications. Ontology for the Sisyphus elevator design problem (VT) is described in [ST99]. In KACTUS project, ontologies for diagnosis of electrical networks and for the exchange of knowledge about ship design and oil platforms have been written [LAB+96]. YMIR ontology is a domain independent, sharable ontology for the formal representation of engineering design knowledge, based on systems theory [ALB93]. PHYSSYS ontology is less biased to a mathematical representation [BAT97]. Knowledge formalized in PHYSSYS has been used to develop number of applications such as a model revision assistant, OLMECO library of model fragments for simulation and for ecological product disassembly analysis. EngMath is an ontology used for mathematical modelling in engineering applications [GRU92]. It has been reused many times such as in PHYSSYS and in CML. CML is an ontology about time, continuity and object properties to enable the sharing of models based on compositional modelling [FFB+94, FF91 FOR84]. In addition, CML ontology has been used to develop ontology for thermodynamic systems and ontology for VT.

Penman Upper Model is a general model about natural language that can be used for the generation and processing of different languages such as Italian, German and English [BMR94]. Other ontologies formalize the semantics of the part–whole relation in natural language [GUA04]. Also, natural language about movement in the French language has been formalized [SAB93]. Ontologies are also used for the development of systems for extraction of knowledge [VRS99].
Corporate and Enterprise Modelling

TOVE ontology formalizes knowledge about production/communication processes, activities, causality, resources, quality and cost in business enterprises [FCF93, GS02]. Ontologies have also been developed for the implementation of knowledge bases for formalization and conservation of the knowledge of experts in enterprises such as KONE ontology that deals with conservation of corporate knowledge about crankshaft design [GRU92].

Medical Diagnosis and Knowledge Acquisition

Knowledge in the medical domain about diagnosis, therapy planning and patient monitoring has been formalized in the GAMES-II project [FS94, HSW+97]. As ontology formally specifies meta-level domain knowledge, it can be an excellent specification for tools that acquire knowledge from domain experts such as PROT ´EG´E-II project [FGD92].

2.2.2 Specification and Conceptualization

We have observed that, desirable qualities for ontologies such as being decomposable, extensible, maintainable, modular and interfacable tied to the information analyzed, universally understood, and translatable characteristics [FGJ97, GUA04]. Also, these are desirable for interoperable software components or classes of objects in object-oriented design [KG02, GFV96, SMJ02, WAR09]. Practitioners from other fields may use different terminology but its meanings are often similar. Hence, Figure 2.2 shows aspects of two general disciplines that can help develop ontologies at the specification and conceptualization stage namely; Modelling and Metamodelling [AW06, GL02]. In practice, knowledge of these disciplines helps to organize the knowledge acquisition process. Moreover, it specifies the ontology’s primary objective, purpose and scope. Lastly, it builds initial vocabulary and organizes taxonomy in an informal or semiformal way and possibly using an intermediate representation [SBF98, MS00, RL02, VSS+05].
Modelling

Ontologies are specific, high-level models of knowledge underlying all things, concepts, and phenomena [GS02, GHW02]. While with other models, ontologies not represent the entire world of interest. Rather, ontology designers select aspects of reality relevant to task [CJB98, VRM+99]. For example in domain of books, ontology designer selects one set of book attributes when developing ontology of a library, and different set when developing ontology of bookbinding. All models follow principles and constraints termed as concept relations and axioms [SMJ02]. Although, there exist different ways to represent ontologies such as ontological engineers most frequently use hierarchical modelling at conceptualization level [DR99, LEN95]. These ontologies represent concept hierarchies and taxonomies in layers and use pictorial representation to visually enhance representation [FGD92]. Layers in ontology representation range from domain-independent to task and domain-specific. As a result, ontologies contain knowledge of appropriate hierarchical and or layered models of relevant world [GFV96, GF95].

Metamodelling

Conceptualizing and specifying ontologies have a strong metamodelling essence [GS02, GHW02]. A metamodel of a modelling technique, improves the rigor of different but similar models [FS97]. Ontologies accomplish same for knowledge models. Without ontologies, knowledge bases representing knowledge of the same domain are generally incompatible while using similar knowledge models [CAC01, HO098, NFF+91]. Metamodelling is preferred because it allows practitioners to preserve the usefulness of any specific model. Ontology simply provides the skeleton for the corresponding models of the domain knowledge [GFV96, GF95]. Generally, ontology is a metamodel describes a way to build models. Its components such as the concepts it defines and the relations between them are always (re)used as building blocks when modelling the parts of domain knowledge [MF03, TL00, SMB07]. When developing a practical software system, it helps the mechanism that with built-in knowledge of the models to be deployed and makes the development mechanism intelligent [KL02].
Many potentially useful parallels exist between ontological engineering and software engineering disciplines such as software architectures and software patterns [GF95, SS99]. It is observed that, many practitioners understand similarities between phases of the ontology development and software development processes [EW05, WAR09]. Thus, there exist potential scope of research and software practitioners can benefit from knowing more about such useful parallels. To accomplish this, software practitioners must seize a range of goals with the help of ontological engineering.

2.3 Goals for Software Practitioners

For software practitioners, ontology based development is extremely important to attain software knowledge acquisition, exchange and reuse [BAR06, DEV02, GUA98, CJB99]. However, ontological engineering enable achieving a range of goals such as to precisely define terms and highly structured definitions of domain concepts, not text-based information. Next, it provides consensus knowledge of a community of people and high expressiveness. Then, coherence and interoperability of resulting knowledge bases is made available by it. In addition, it endows with stability and scalability of ontologies. Lastly, it organizes a foundation for solving a variety of problems and constructing multiple applications.

Although, ontologies are content-related than representation-related and achieving these goals calls for formalization and co-existence of artistic creativity and systematically applied knowledge from other disciplines [DW99, FCM+03]. Ontology can be developed collaboratively by many distributed individuals and organizations with differing expertise, goals, and interactions. Various communities of experts and practitioners examine problems from different perception and are concerned with different dimensions of the content’s semantics and representation [MA05, NM04, SSS+01]. These individuals need to properly understand each other and meaningfully communicate their views of domain knowledge to form meaningful higher-level knowledge [OVR+06, TA00].
Once application developers are ready to use the ontology, developers should be able to convert it into a desired form, such as object base or knowledge base, using representation methods [GRU92]. Hence, ontological engineering must rely on several content formats, frameworks, and development strategies that reduce semantic ambiguity and allow for sharing and reusing knowledge and practices from other disciplines. In addition, ontological engineering involves developing higher-level knowledge-based products that express the consensus knowledge of a community of agents [SMB07, MC06, CHK+07]. Certain software engineering disciplines and issues rarely discussed by ontology researchers that can help software engineers and practitioners. These include knowledge intensive information system modelling, software requirement engineering, software reuse, software reliability and software security and abstraction, as presented in Figure 2.3.

2.4 Software Engineering Disciplines

Software developers consider ontologies as a trend involving methodology and technology as AI community develops ontologies that use special-purpose mechanisms [SMJ02, WAR09]. However, ontology is always about entities and relationships. In addition, methodology from traditional software engineering such as using ER model, top-down decomposition strategy and structured system analysis are used to represent it. For example, Methontology framework for developing ontologies proposes a close relative of traditional waterfall model of software development for an ontology development lifecycle [FGJ97, LGS+99]. Moreover, entire ontology developed using Methontology framework is stored in a relational database and can encode its ontology in its data dictionary [MIZ98, FH97]. All design criteria for ontologies, such as clarity, extensibility, coherence, and minimal encoding bias also represent design criteria for software systems modules [FCM+03, GF95]. Ontology researchers and developers can explore a large
variety of iterative and incremental traditional software development methodologies for new ideas in ontology based software engineering. We consider some of them and discuss as follows:

2.4.1 Ontology Driven Information System

Ontology development process nearly coincide with those of object-oriented software development [LGS+99, MIZ98, VRM+99]. In both cases, it is important to assemble domain vocabulary in the beginning, often starting from the domain’s generic nouns, verbs, and adjectives [FS97, ST99, GFV96, GF95]. Object-oriented analysis stresses different aspects than ontological analysis and yet analogous [MIZ98, JAC92, SMJ02]. The result of object-oriented analysis is a draft of domain ontology relevant to the application. Besides, as object-oriented designers define classes, objects, hierarchies, interface functions and system behavior, ontological engineers use intermediate representations such as semantic networks, graphs, and tables to design hierarchies and other concept relationships [CAC01, FEA+02]. Both types of specialists use templates to specify product details [FS97, LGS+99]. Classes can be merged or refined with ontologies. Class libraries and previous design specifications often provide reuse in object-oriented design with the help of previously encoded and available ontologies [FMR98, JCJ+07, WAR09].

In addition, important differences exist between Ontology Development Life Cycle (ODLC) and Object Oriented Software Development Life Cycle (OOSDLC) from practitioner’s perspective. ODLC signifies knowledge-level stance in describing system, while OOSDLC largely refers to the means of design and implementation [CJB99, H0098]. For example, in semantic-based information retrieval system, ontologies specify the meaning of concepts to be searched, while object-oriented design represents domain models [KL02, MS00]. Object-oriented design languages such as UML offers explicit design methodology and notation for all design artifacts, but ontological and metamodelling principles are only implicit in those languages [MIZ98, BKK+02]. In other words, ontology is abstracted at knowledge level from corresponding class diagrams, object diagrams, and use-case diagrams, represented in any
object-oriented notation such as UML [SBF98, WER+97, SK03]. The role of ontology is to convey and explicitly specify domain concepts, terms, definitions, relations, constraints, and other semantic contents that object-oriented analysis and design should rely on and support [EMB04, GUA04, MC06, SS06, SK04].

It is observed that one area of ontology based software engineering requires additional efforts involves developing a generally accepted notation for representing ontologies. Software engineers have used several different notations in object-oriented design over the past decade, but all have converged to UML notation, which provides a metamodel of object-oriented design. It defines graphical notation for representing classes, objects, and relationships, covering all practical aspects of object oriented design [JAC92, JCJ+07]. But, ontological engineering strive standard notation that is accepted, understood, and used in practice [VRM+99 FS97, WAR09].

2.4.2 Software Requirement Engineering

The basics of ODLC involve designing and specifying overall system structure and underlying organization [GUA98, GUA04, HOO98]. Ontologies are architectural armatures for building knowledge bases, models, and software structural designs [SBF98, ST99]. This assertion helps requirement engineers to build requirement specifications of any application. Structural design style in the field of requirement engineering, characterizes a family of systems related by shared structural and semantic properties [SHA95, LV02, LV04]. A style typically defines a vocabulary of design elements, design rules for compositions of elements and semantic interpretation of design element compositions [VAN03]. Many successful designs can share a style. Styles contain condensed skeletons of the architectural knowledge gained by experienced software designers, and provide a means to reuse that knowledge [BOE96, BOS95, LPR93].

In addition, ontologies structure knowledge in form of layers to separate use-specific knowledge from more reusable knowledge [MIZ98, VRM+99]. Other structural design styles help to define requirement engineering solutions include
pipeline and data abstraction [SHA95, BL03, EW05]. In addition, layered style is suitable for applications involving distinct classes of services that can be arranged hierarchically. It is explored that, researchers have proposed layers for building requirements by considering basic system level services and utilities appropriate to many applications [ST99, SG05, SPL06, ZZY+07]. But, it lacks in specific application task that depends on requirement type to make knowledge intensive requirement engineering process.

2.4.3 Software Reuse

Early software reuse practices focused on code and made ad hoc. However, reuse changed as the industry matured. Reuse became planned and systematic [ABH+99, DEV02, GUA98]. Currently, any product of the software life cycle can potentially be reused. According to some researchers the active areas of reuse research includes reuse libraries, domain engineering methods and tools, reuse design, design patterns, domain specific software architecture, component, generators, measurement and experimentation [MA05, HAM04, NM01, RL02]. However, ideas emerging from this period lacks in reuse design principles, commonality and variability analysis, and various approaches to knowledge generators [FK05].

A significant objective of ontology is to build reusable knowledge components and knowledge-based services that can be invoked over networks [SZY98, REI97, MVI95, SS99]. Consequently, software engineering field is attempting to develop repositories of reusable, pretested, interoperable, and independently upgradable software components that enable plug-and-play design. These objectives necessitate designing systems from application elements constructed independently by developers using different languages, tools, and computing platforms [SMB07, WZX06]. Ontologies can precisely define the semantics of components as well as the types of relations and communication between software components [HAM04, MERM03, LINK6]. Consequently, ontologies are used to enable a basis for designing and developing interoperable software components in practice.
2.4.4 Software Reliability

Software reliability is probability of failure free operation of a computer program in a specified environment for a specified time [FH97, LEN95]. Software reliability has been discussed, in a number of studies on software reliability evaluation focusing on post-software development that includes reliability modelling, reliability estimation and tools development [LGS99, MIZ98, SHA95]. Reliability evaluation taking place prior to software development is attracting a growing attention among software architects and reliability experts. This issue tackles by introducing an ontology-based approach. This approach is characterized with integration of software reliability engineering and software architecture design [GHJ+95, LEN95, SZY98]. In particular, this approach suggests software architecture design as the first phase of evaluating reliability in the development of software systems [VRM+99]. However, reliability can be accomplished at requirements analysis phase.

2.4.5 Software Security

Security analysis and design involves the identification of security attributes and the design of solutions that address these attributes in an efficient and effective manner [MGM03]. Effective software security control has been emphasized mainly to ontology based projects due to its expediency, flexibility and comprehensibility. Consequently, it needs methodology of improving the current posture of project security while developing these projects with various perspectives [SI11a, SI11b, SI11d]. It attempts to provide a range of benefits related to ontology based projects. It is observed that these benefits may incur different unvisualized states and for ensnaring these states, ontology based software projects thereby indulge with the instinct for security attributes. Conversely, the involvement of these factors may be horded in such a way that may render to acquire the security perspective of ontology based software projects.
2.5 Summary

In this chapter, we have highlighted the importance of ontologies in software development process while presenting a literature survey component of the thesis. We present ontological engineering in the context of other disciplines and observe that it enables both ontological engineers and other specialists to view respective fields from different perspectives. In this view, goals of software practitioners are covered to consider awareness of such similarities. In addition, these enable to create new ways to build and improve knowledge. Chapter proceeds with introduction of various software engineering disciplines: information system building, requirement engineering, reuse, reliability and security. We observe that, ontologies are needed in all software engineering disciplines to explore entities, attributes and relationships in the relevant world. Moreover, all disciplines require knowledge that constitutes data structures, methods, or algorithms.