2015 International Conference on Computational Science and Computational Intelligence

Towards an Ontology Design Architecture

Jason Jaskolka*
St. Francis Xavier University
Antigonish, Nova Scotia, Canada
Email: jjaskolk@stfx.ca

Wendy MacCaull
St. Francis Xavier University
Antigonish, Nova Scotia, Canada
Email: wmaccaul@stfx.ca

Ridha Khedri
McMaster University
Hamilton, Ontario, Canada
Email: khedri@mcmaster.ca

***Abstract*—As the world enters the age of "big data", new ways to represent and reason on enormous amounts of data are demanded and expected. Work in developing ontologies and reasoning approaches have taken steps towards addressing these needs. However, ontology development is not usually perceived as an engineering activity. Developers often overlook fundamental questions and concerns of ontology design and an ad hoc "one-time use" mentality has emerged. In this position paper, we address this lack of design consideration in ontology development by adapting existing software design patterns. We propose an ontology design architecture based on the model-view-controller (MVC-II) architectural style in an effort to present a new engineering view of designing ontologies.**

***Keywords*-ontology design, ontology engineering, MVC-II architecture, knowledge representation, separation of concerns.**

I. Introduction and Motivation

Today's world is filled with a plethora of information and new and updated ways to represent and reason on vast amounts of data are required. The emergence of ontologies and reasoning approaches has led to improvements in the ability to reason on large amounts of information. However, appropriately designing and developing ontologies suitable for the emerging reasoning needs in the age of "big data" still faces many difficulties. There does not exist a set of agreed-upon guidelines and methods for designing and developing ontologies [1]. Developers often make leaps from knowledge acquisition phases straight to implementation phases, often overlooking fundamental questions and concerns of ontology design. Consequently, developers quickly reach a number of roadblocks in their quest to develop ontologies that are modifiable, extendable, and reusable [2].

Throughout this paper, an *ontology* is perceived as a complete system that provides an understanding of a world. Based on this understanding, we propose an ontology design architecture based on a variation of the existing model-view-controller (MVC-II) style inspired from the area of software engineering. The architecture aims to provide a clear separation of concerns with respect to the knowledge representation and the reasoning abilities of the ontologies developed. It also looks to provide a separation of the domain-independent and domain-specific knowledge required of an ontology to capture particular views of the possible worlds that it needs to consider. Overall, our architecture provides a systematic way to guide ontology design and development.

We do not claim to provide a new methodology for developing ontologies from start to finish, or that existing methodologies for developing ontologies are not important. Rather, we target a framework that supplements the ontology development phases of existing methodologies. The views in this paper intend to break down the current convention of developing ontologies in an ad hoc manner without sufficient assessment of the questions and concerns required of designing an ontology for practical use and reuse.

This paper is organised as follows. Section II discusses the related work. Section III presents the proposed ontology design architecture. Section IV discusses the benefits and drawbacks of the proposed architecture. Section V concludes and gives the highlights our current and future work.

II. Related Work

A. Methodologies for Ontology Development

Developing an ontology from scratch has largely been considered more of an art rather than an engineering activity [3]. Different groups build ontologies with a variety of different approaches, methods, and techniques. There is no universal agreement on the overall landscape of existing ontologies [4]. There is a lack of standardisation with regard to the activities, life-cycles, methodologies, and sets of well-defined design criteria, techniques, and tools to realise the development of ontologies as an engineering activity [1], [3].

The characterisation of the ontology development life-cycle has received much attention in the past. A detailed summary of existing methodologies can be found in [3]. Of these, METHONTOLOGY [5] has become the most prevalent. While METHONTOLOGY outlines many of the phases required when developing an ontology, it is missing the notion of a design phase. By omitting design phases, resulting ontologies are often poorly thought-out in terms of their maintainability, modifiability, extendability, and reusability. As with the development of any other engineering system, a proper design phase is required in the ontology development life-cycle to ensure that a developed ontology is fit for purpose, that it meets its requirements and objectives, and that it exhibits a set of desirable quality attributes.

*Corresponding Author Current Address: *McMaster University, Hamilton, Ontario, Canada, Email: jaskolj@mcmaster.ca*

978-1-4673-9795-7/15 $31.00 © 2015 IEEE
DOI 10.1109/CSCI.2015.48

B. Archetypes

The idea of archetypes has gained popularity in the healthcare domain, particularly with the rise of electronic health records and the advent of the openEHR framework [6]. Within the healthcare domain, an archetype refers to a detailed and domain-specific definition of a clinical concept in the form of structured and constrained combinations of the data entities, such as *blood pressure* [7]. However, there does not appear to be any reason why the notion of archetypes should be limited to the healthcare domain. If we consider the idea of archetypes in a more domain-independent context, then an *archetype* refers to a knowledge-level model that defines valid information structures [8]. In this way, archetypes can offer general and reusable terminologies that can be adapted to many domains. In this paper, we use the idea of archetypes to provide characterisations of general concepts and the attributes most commonly associated with them. For instance, a Person archetype may specify the general concept of a person with the attributes: *name*, *address*, and *phone number*. We assume this understanding of the term *archetype* for the rest of this paper.

C. Ontology Design Patterns

An *ontology design pattern* is a reusable successful solution to a recurrent modelling problem (e.g., [9], [10]). As such, it serves the same purpose as design patterns in other fields of engineering where the intention is to provide modular, reusable, and replaceable building blocks for larger systems. While much research into developing ontology design patterns has been done in recent years, there is yet to be a wide adoption of the design pattern approach by practitioners, largely due to the poor documentation and large number of proposed ontology design patterns [10]. Consequently, it is often difficult for a practitioner to select and adapt a design pattern that models the concepts and phenomena that are relevant to their needs. The question of whether there is a need for new design patterns specific to ontologies arises. To the best of our knowledge, there does not appear to be any evidence against adapting the current widely-used engineering design patterns, such as those found in the software engineering field, for ontologies.

III. The Proposed Architecture

We propose to design and develop an ontology from an engineering perspective as one would approach the design and development of any other engineering system, such as a bridge, a building, or a software system. Our architecture is based on a variation of the MVC-II architectural style adapted from the area of software engineering.

MVC-II is a variant of the model-view-controller (MVC) architecture where the controller and view components are separated. It is best suited for interactive applications where multiple views are required for a single data model and where its interfaces are prone to frequent changes [11].

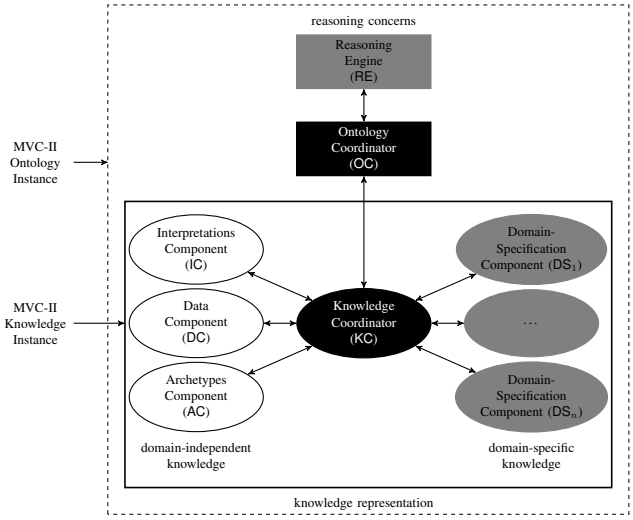

Figure 1. A nested MVC-II architecture for ontology design

The MVC-II architectural style consists of three primary components. The *model* (white components in Figure 1) provides all of the core functional services and encapsulates all data details, independent of the other components in the system. The *view* (grey components in Figure 1) provides particular views of the *model* component. The *controller* (black components in Figure 1) manages all of the initialisation, instantiation, and registration of the other system components and is responsible for selecting desired views and managing user input requests. Because an ontology can be seen as an interactive system where multiple views of the knowledge representation are required in order to complete reasoning tasks, it fits the application domain of the MVC-II architectural style. Due to its modularity, MVC-II can offer enhanced modifiability, extendability, and maintainability.

We propose a *nested MVC-II architecture*, shown in Figure 1. It consists of two instances of the MVC-II architectural style. The inner *Knowledge Instance* (elliptical components), provides an MVC-II-based architecture for the knowledge representation of the ontology and offers separation between domain-independent knowledge and domain-specific knowledge. The outer *Ontology Instance* (rectangular components), provides an MVC-II-based architecture offering separation between the knowledge representation and the reasoning concerns of the ontology.

A. The Knowledge Instance

The *Knowledge Instance model component* encompasses the domain-independent knowledge representation of the ontology and consists of the *Archetypes Component*, *Data Component*, and *Interpretations Component*.

The *Archetypes Component* (AC) is concerned with identifying the set of concepts and their respective attributes required to model the world dictated by the requirements of the ontology being developed. AC is related to identifying and/or developing suitable *archetypes* that capture

the general structure of the conceptual knowledge of the world being be modelled. This involves specifying the types of attributes that describe each concept. For example, an ontology related to students and grades may contain a Person archetype with attributes *name* : PersonName and *id* : Integer, a PersonName archetype with attributes *first* : String and *last* : String, and a Grade archetype with an attribute *grade* : GradeType. AC is suitable for use or reuse in many domains as the archetypes for the required concepts are persistent in all possible worlds.

The *Data Component* (DC) is concerned with the data provided by the problem and the requirements of the ontology being developed. It provides a collection of facts by identifying instances of the concepts, and their attributes, provided by AC. One can think of DC as an interface to the data sources providing the facts for the given problem. An example data source may be a table of grades achieved by a student. The idea is to have the ability to "plug-in" new data sources as they become available or as they are needed. It is through these data sources that the concepts defined in AC and each *Domain-Specification Component* (DS) can be instantiated with assertions about the domain of interest.

The *Interpretations Component* (IC) is concerned with providing concrete interpretations for the abstract types that are specified in AC and instantiated by DC. For example, the abstract GradeType identified from the above mentioned Grade archetype can have different interpretations. A grade can be represented as a *letter-grade* (e.g., B+), as a *percentage* (e.g., 85%), or as a *grade-point-average* normalised to some standard (e.g., 3.5/4.0). IC allows the ontology being developed to contain concepts and relationships that have different embodiments in different possible worlds which is not currently considered in the literature. It allows for different independent interpretations to be considered when reasoning on the knowledge contained within the ontology.

The *Knowledge Instance view component* consists of a collection of *Domain-Specification Components* ($\mathsf{DS}_1, \ldots, \mathsf{DS}_n$), each of which is concerned with providing domain-specific knowledge, giving specific viewpoints of the domain-independent knowledge contained within AC, DC, and IC. Each viewpoint is encapsulated in a single DS responsible for providing the interpretations of concepts and relationships within its specific viewpoint. Moreover, each DS is concerned with identifying specialisations of the archetypal concepts identified in AC. For example, in a domain involving students and grades, we can identify a student viewpoint as a specialisation of the Person archetype from AC. The viewpoint will inherit the attributes specified by the Person archetype and will extend it with domain-specific attributes. For instance, a Student viewpoint can be specified with attributes *student* : Person and *gradesEarned* : ItemsType⟨Grade⟩ where the abstract ItemsType will have a concrete interpretation, provided by IC, such as *Set*, *List*, or *Bag*.

Lastly, the *Knowledge Instance controller component* consists of the *Knowledge Coordinator* (KC) which is concerned with managing and coordinating the domain-independent knowledge contained within AC, DC, and IC, and the domain-specific knowledge contained within each DS. As such, KC maintains a registry of the possible concepts, data, and interpretations, as well as possible domains through the registration and initialisation of the other components in the *Knowledge Instance*. KC allows for the ability to state that *Joe Smith* is a Student and has earned a Grade of *76%* which is interpreted as a *percentage*, for example.

B. The Ontology Instance

The *Ontology Instance model component* provides the *knowledge representation* of the ontology being developed and is comprised of the *Knowledge Instance* (denoted by the elliptical components at the bottom of Figure 1).

The *Ontology Instance view component* consists of the *Reasoning Engine* (RE) which is responsible for interfacing with existing reasoning tools and for defining particular understandings of the world in which to reason about, in order to answer questions posed to the ontology. It provides the capability to set up reasoning tasks, based on the input of the user and the requirements of the ontology, to allow for different configurations of the knowledge representation in the form of different interpretations and/or viewpoints in order to reason on multiple possible worlds. Furthermore, RE admits the specification of knowledge and information management approaches that ought to be considered in order to address issues of inconsistent or conflicting information contained within the ontology knowledge representation.

Finally, the *Ontology Instance controller component* is comprised of the *Ontology Coordinator* (OC) which encompasses the initialisation, instantiation, registration, and coordination of the *Ontology Instance model* and *view components* to facilitate the interaction between the reasoning tasks and the knowledge representation for the ontology being developed. OC is the main controller responsible for selecting the appropriate domain-specific viewpoints, concrete interpretations, data sources, etc., that are required to answer the questions from RE. For instance, in a domain of students and grades, consider the question: "Which student has the highest average grade?" For this example, suppose that the user indicates that it only wishes to consider the *Bag* interpretation offered by IC for the collection of grades. In this case, OC is responsible for handling the user input passed from RE in order to decide how to communicate the requirements of the reasoning task to KC so that the student domain-specific viewpoint, and the *Bag* interpretation of the collection of each student's grades can be selected and used to find the name of the Student with the maximum average grade using the data provided by DC. In simple terms, OC provides the bridge between the knowledge representation and the reasoning concerns of the developed ontology.

IV. Assessment of the Proposed Architecture

A. Benefits of the Proposed Architecture

There may be a number of different ways in which to design an ontology to meet its requirements. However, by following the proposed architectural framework, we conjecture that the developed ontologies will have a number of beneficial qualities. Specifically, the proposed architecture provides a separation of concerns at multiple levels. First, the *Ontology Instance* separates the knowledge representation and the reasoning concerns of the ontology being developed. Second, the *Knowledge Instance* separates the domain-independent and domain-specific knowledge, enabling the relatively stable domain-independent knowledge to be designed and developed independent of the more volatile domain-specific knowledge. Finally, the *Knowledge Instance view component* separates the different possible viewpoints of the domain-independent knowledge. As a result, developed ontologies can exhibit enhanced modifiability, extendability, and maintainability. When a modification or extension is required, a developer only needs to identify the concern of the proposed changes in order to locate which components need to be modified or extended. This enables the developed ontologies to be maintained over an extended period of time. Also, the developed ontologies can benefit from enhanced reusability. For example, AC, DC, and IC can be reused in a variety of application domains driven by the requirements and context of the ontology to be developed. Moreover, the knowledge representation (*Ontology Instance model component*) of the developed ontologies can be reused to address different reasoning concerns and to answer different questions in the domain of interest. Lastly, the proposed architecture can be extended to enable many concurrent KCs, by extending the MVC architecture into a PAC architecture. The latter is developed from MVC to support multitasking and concurrency. Therefore, it is straightforward to take the proposed architecture and extend it to handle large data sets.

B. Drawbacks of the Proposed Architecture

The drawbacks of the proposed architecture are related to the amount of communication overhead and the complexity of the controller components. The design of KC and OC is a tricky problem. Much effort is required in order to handle the communication necessary to coordinate the other components in the system. This additional controller complexity is an inherent drawback of the use of the MVC-II architectural style [11]. Much care needs to be taken in designing the controller components to ensure that the correct knowledge and information is available to, and communicated by, KC and OC so that queries can be answered properly. Poorly designed controllers can effectively render the system unusable and is a problem that must be investigated. However, it should be noted that controller components are relatively stable and are not prone to frequent changes.

V. Conclusion and Future Work

We proposed an ontology design architecture that adopts a nested MVC-II-based architectural style supporting separation of concerns. Ontologies designed using the proposed architecture can benefit from enhanced modifiability, extendibility, and reusability, which addresses some criticisms of the state-of-the-art and helps to eliminate the current ad hoc "one-time use" mentality of ontology development.

The proposed architectural framework can be adopted to systemically design ontologies in a structured way. This leads to a more refined engineering approach to ontology development. We have further illustrated the usage and benefits of the proposed architecture, including support for concurrent and distributed reasoning, in [12]. However, the practical use of the proposed architectural framework to design real-world ontologies requires more study. A user study is needed to evaluate the effectiveness and impact of the proposed architecture on ontology development.

Acknowledgment

This research is supported by the Natural Sciences and Engineering Research Council of Canada through the grant RGPIN-2014-06115.

References

[1] V. Devedžić, "Understanding ontological engineering," *Comm. of the ACM*, vol. 45, no. 4, pp. 136–144, Apr. 2002.

[2] O. Corcho, "10 basic rules to overcome ontology engineering deadlocks in collaborative ontology engineering tasks," Ontology Summit 2014 Track-C: Overcoming Ontology Engineering Bottlenecks - II, Mar. 2014.

[3] A. Gómez-Pérez, "Ontological engineering: A state of the art," *Expert Update*, vol. 2, no. 3, pp. 33–43, 1999.

[4] J. Geller, Y. Perl, and J. Lee, "Ontology challenges: A thumbnail historical perspective," *Knowledge and Information Systems*, vol. 6, no. 4, pp. 375–379, 2004.

[5] M. Fernández, A. Gómez-Pérez, and N. Juristo, "METHONTOLOGY: From ontological art towards ontological engineering," in *Proc. of the AAAI Spring Symposium on Ontological Engineering*. AAAI Press, 1997, pp. 33–40.

[6] openEHR Foundation, "What is openEHR?" Available: http://www.openehr.org/what_is_openehr (June 25, 2015), 2015.

[7] C. Martínez-Costa, M. Menárguez-Tortosa, and J. T. Fernández-Breis, "An approach for the semantic interoperability of ISO EN 13606 and OpenEHR archetypes," *Journal of Biomedical Informatics*, vol. 43, no. 5, pp. 736–746, 2010.

[8] T. Beale, "Archetypes: Constraint-based domain models for future-proof information systems," in *Proc. of the 11th OOPSLA Workshop on Behavioral Semantics: Serving the Customer*, K. Baclawski and H. Kilov, Eds., 2002, pp. 16–32.

[9] A. Gangemi and V. Presutti, *Handbook on Ontologies*, 2nd ed. Springer, 2009, ch. Ontology Design Patterns.

[10] K. Hammar, "Ontology design patterns: Adoption challenges and solutions," in *Proc. of WaSABi 2014*, vol. 1240, 2014.

[11] K. Qian, *Software Architecture and Design Illuminated*. Jones & Bartlett Learning, 2010.

[12] J. Jaskolka, W. MacCaull, and R. Khedri, "Towards an architectural framework for systematically designing ontologies," McMaster University, Tech. Rep. CAS-15-09-RK, Nov. 2015.