Department of Computing and Software

Faculty of Engineering — McMaster University

Towards an Architectural Framework for Systematically Designing Ontologies

by

Jason Jaskolka, Wendy MacCaull, and Ridha Khedri

CAS Report Series **CAS-15-09-RK**

Department of Computing and Software **November 2015**
Information Technology Building
McMaster University
1280 Main Street West Hamilton, Ontario, Canada L8S 4K1

Copyright © 2015

Towards an Architectural Framework for Systematically Designing Ontologies

Jason Jaskolka[1,2], Wendy MacCaull[1], and Ridha Khedri[2]

[1] Department of Mathematics, Statistics and Computer Science, Faculty of Science, St. Francis Xavier University, Antigonish, Nova Scotia, Canada

[2] Department of Computing and Software, Faculty of Engineering, McMaster University, Hamilton, Ontario, Canada

Technical Report CAS-15-09-RK
Department of Computing and Software
McMaster University

November 2, 2015

Abstract

Over the last decade, the world has entered into the age of "big data" and new ways to represent and reason on enormous amounts of data are required. Work in developing ontologies and sophisticated reasoning approaches partly addresses these needs. However, ontology development is currently more of an art than an engineering activity. Fundamental questions and concerns of ontology design are often overlooked by developers and an ad hoc "one-time use" mentality has emerged. Consequently, resulting ontologies are often difficult to modify, extend, and reuse. To address this lack of design consideration in the ontology development life-cycle, we propose an architecture based on a variant of the model-view-controller (MVC-II) architectural style, thereby giving a new engineering view for ontology design. The proposed architecture provides a step towards an ontology design framework that can supplement existing development methodologies.

Keywords: ontology design, ontology engineering, MVC-II architecture, knowledge representation, reasoning, separation of concerns

Contents

1 Introduction and Motivation **1**

2 Related Work **2**
2.1 Existing Methodologies for Ontology Development 2
2.2 Archetypes 3
2.3 Ontology Design Patterns 3

3 An Ontology Design Architecture **4**
3.1 An Example Ontology Design Scenario 5
3.2 The Knowledge Instance of the Proposed Architecture 6
3.2.1 Archetypes Component 6
3.2.2 Data Component 6
3.2.3 Interpretations Component 8
3.2.4 Domain-Specification Components 8
3.2.5 Knowledge Coordinator 9
3.3 The Ontology Instance of the Proposed Architecture 10
3.3.1 Reasoning Engine 10
3.3.2 Ontology Coordinator 11

4 Merit of the Proposed Ontology Design Architecture **11**
4.1 Benefits of the Proposed Architecture 11
4.2 Drawbacks of the Proposed Architecture 12

5 Designing an Ontology Using the Proposed Architecture **13**
5.1 Designing the Knowledge Instance of the Proposed Architecture 13
5.1.1 Designing the Archetypes, Data, and Interpretations Components 13
5.1.2 Designing the Domain-Specification Components 14
5.1.3 Guidelines for Designing the Knowledge Coordinator 15
5.2 Designing the Ontology Instance of the Proposed Architecture 15
5.2.1 Guidelines for Designing the Reasoning Engine 16
5.2.2 Guidelines for Designing the Ontology Coordinator 16
5.3 Discussion 17
5.3.1 Support for Modifiability 17
5.3.2 Support for Extendability 17
5.3.3 Support for Reusability 19
5.3.4 Support for Collaborative/Parallel Development 19
5.3.5 Support for Concurrent and Distributed Reasoning 20

6 Conclusion and Future Work **22**

1 Introduction and Motivation

As the world continues to push towards new technologies and as the amount of information that is collected and transferred continues to grow, new ways to represent and reason on copious amounts of data are required. The reasoning needs of today's world are not the same as they once were with the development of expert systems in the 1980s. The old classification systems [50] are now outdated. With such enormous amounts of data, there is an evolving need for *concurrent* and *distributed reasoning* that gives the ability to reason about different parts of information or on different concerns or viewpoints in parallel.

The emergence of ontologies and sophisticated reasoning approaches has led to improvements in the ability to reason on large amounts of information. However, there are still many difficulties in appropriately developing ontologies that can be used for the emerging reasoning needs in the age of "big data". The process of developing an ontology is more of an art rather than an engineering activity [21]. It is often the case that development teams follow their own set of development phases and design principles and criteria. Because of this, there does not exist a set of agreed-upon guidelines and methods for designing and developing ontologies [14]. This leads to development teams making leaps from knowledge acquisition phases straight to implementation phases, often overlooking fundamental questions and concerns of ontology design. Consequently, development teams quickly reach a number of roadblocks in their quest to develop ontologies that are modifiable, extendable, and reusable (see, for example, [10]).

Perhaps the most significant issue plaguing the development of ontologies is the sparse availability of methodologies that seriously consider design concerns [39]. Geller et al. [19] suggested that this problem can be addressed by establishing and using better methodologies when developing ontologies from scratch, or from smaller ontologies. The standardisation and adoption of a generally accepted methodology and notation for designing and developing ontologies is needed [14, 21]. Such a methodology that everyone accepted, understood, and used in practice would undoubtedly increase the modifiability, extendability, and reusability of ontologies. Furthermore, it is well-known that at the heart of many ontologies is the need to specify knowledge and to automatically draw conclusions or information from the knowledge and data sets. As such, related investigations into ontology development aim to tackle the well-known trade-off between semantic expressivity and computational feasibility. However, due to the distributed nature of most of their applications, ontologies require the integration of multiple knowledge sources and the need to relate heterogeneous ontological specifications [27]. Although there has been some work in the area of concurrent and distributed reasoning (e.g., [2, 42]), the current state-of-the-art for ontology development (e.g., [11, 15, 21, 44]) largely reflects a perspective that fails to recognise its need in many modern application domains.

Throughout this paper, an *ontology* is viewed as a complete system that gives an understanding of a world. With this understanding, we propose an ontology design architecture based on a variation of the model-view-controller (MVC-II) architectural style in an effort to address the lack of design effort put forth in current ontology development life-cycles. This use of the MVC-II architectural style is inspired from the area of software engineering. The proposed architecture aims to provide a clear separation of concerns with respect to the knowledge representation and reasoning abilities of the developed ontologies. Furthermore, it looks to separate the domain-independent and domain-specific knowledge required of an ontology to capture particular viewpoints of the possible worlds that it needs to consider.

We do not claim to provide a new methodology for developing ontologies from start to finish, or that existing methodologies are not important. Rather, we target a framework that supplements the development phases in existing methodologies. We seek to enrich current development methodologies by injecting a more systematic and dedicated design phase into the currently established life-cycle. Our goal is to offer an architectural design framework that practitioners can adopt in order to develop ontologies that are modifiable, extendable, and reusable, and that also fit a particular purpose. It should also be noted that we do not claim to solve every issue with regard to ontology design. Instead, 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 for a design that permits practical use and reuse. We therefore propose to establish a new engineering view of ontology design and development. In this paper, we take a few steps in this new direction and we indicate what the next steps should be.

The remainder of this paper is organised as follows. Section 2 provides a discussion of the related work. Section 3 articulates the proposed ontology design architecture. Section 4 discusses the benefits and drawbacks of the proposed architecture. Section 5 gives an illustrative example to show how to design an ontology using the proposed architecture and to highlight the benefits of the resulting design. Finally, Section 6 gives concluding remarks and points to the highlights of our current and future work.

2 Related Work

In this section, we point to the lack of design considerations in ontology development as motivation for the need for an appropriate design architecture. We survey the literature related to existing methodologies for ontology development, archetypes, and ontology design patterns.

2.1 Existing Methodologies for Ontology Development

The characterisation of the ontology development life-cycle has received much attention in the past [22]. Perhaps one of the first methodologies for building ontologies resulted from the experience in building the *Cyc* knowledge base [28]. Later, Uschold and King [48] proposed a method based on the experience in developing the Enterprise Ontology and Grüninger and Fox [23] proposed a methodology inspired by the construction of knowledge-based systems using first order logic. Next, a methodology based on a bottom-up approach involving the abstraction of an application knowledge base was proposed while working on the KACTUS project [6] and a methodology based on a top-down approach involving the derivation of domain-specific ontologies from large monolithic ontologies resulted from work on the *Sensus* ontology [45]. Then, the On-To-Knowledge [44] and METHONTOLOGY [15, 16] methodologies appeared and have since become the most prevalent. A detailed summary of each of the above mentioned approaches can be found in [11] and [21].

Arguably the most popular methodology for developing ontologies is METHONTOLOGY where the ontology development life-cycle closely resembles a software development life-cycle. *Specification*, *Conceptualisation*, *Integration*, *Implementation* phases are performed sequentially, while *Knowledge Acquisition*, *Evaluation*, and *Documentation* phases are carried out throughout the entire life-cycle. While METHONTOLOGY outlines many of the phases required in ontology development, it is missing the notion of a design phase. Our observation

aligns with what is found in [21] where it is stated that development teams often leap from knowledge acquisition and conceptualisation phases straight to implementation phases. It is well known, in the engineering field, that a good design reduces risks in product development, helps development teams work together in an orderly fashion, and leads to products that have higher quality attributes [41]. 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 development life-cycle to ensure that the developed ontology is fit for purpose, and that it meets its requirements and objectives.

2.2 Archetypes

The notion of archetypes has recently gained popularity in the area of semantic interoperability and in the healthcare domain, particularly with the rise of electronic health records and the advent of the openEHR framework [36]. In the current literature, archetypes are commonly used and discussed within the healthcare domain. As such, in that 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*, *heart rate*, and *diagnosis* [33]. 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 [5]. In this way, archetypes can offer general and reusable terminologies that can be adapted to many domains. For instance, in [29] the notion of archetypes has been adapted to an emergency response domain in order to define emergency concepts such as *tsunami*, *nuclear accident*, and *evacuation mission*. Generally speaking, an archetype can include other archetypes and can be used in combination with one another to define complex conceptual structures[1]. Archetypes enable information systems to guide and validate user input during the creation and modification of information, to guarantee interoperability, and to establish a well-defined basis for efficient querying of complex data.

In this paper, we use the idea of archetypes to provide characterisations of general concepts with the attributes most commonly associated with them. For instance, we can think of a Person archetype that specifies the general concept of a person with attributes *name*, *address*, *phone number*, and *e-mail address*. We may also have a Name archetype which further specifies the general concept of a name with a *first*, *middle* and *last* name, represented as string literals. For the remainder of this paper, we assume this understanding of the term *archetype*.

2.3 Ontology Design Patterns

An *ontology design pattern* is a reusable successful solution to a recurrent modelling problem [7, 18, 24, 25, 40, 43]. 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 the development of ontology design patterns has been done in recent years, there is yet to be a wide adoption of the design pattern approach by practitioners. At the time of writing, there are 172 design patterns submitted to

[1] In the openEHR community, the encapsulation of the combination or inclusion of archetypes is typically called a *template* [35]. However, throughout this paper, we elect to use the term archetype in a more general sense, and we do not introduce new terminology to distinguish this kind of archetype inclusion or combination.

the primary repository for ontology design patterns [34]. To date, not a single ontology design pattern has graduated from the submitted to published status. This is largely due to poor documentation [25]. Many ontology design patterns do not provide adequate descriptions of intents and purpose, consequences of use, or illustrative use cases. Consequently, it is often quite difficult for a practitioner to select and adapt a design pattern that models the concepts and phenomena that are relevant to their needs. Furthermore, some of the currently proposed ontology design patterns appear to be too specific to be widely applicable for different situations. Design patterns aim to provide solutions to classes of problems. When the specificity of the proposed design pattern reduces the class of problems that it solves to a singleton set, the design pattern loses its meaning. 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.

3 An Ontology Design 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. The proposed architecture is based on a variation of the MVC-II architectural style adapted from the area of software engineering and architectural design. MVC-II is a variant of the model-view-controller (MVC) architecture where the view and controller components are separated. It is best suited for interactive applications where multiple viewpoints are required for a single data model and where its interfaces are prone to frequent changes [41]. The MVC-II architectural style consists of three primary components. The *model* is responsible for providing all of the core functional services and for encapsulating all data details, independent of the other components in the system. The *view* is responsible for providing particular viewpoints of the *model*. Finally, the *controller* is responsible for managing all of the initialisation, instantiation, and registration of the other components in the system and is responsible for selecting desired viewpoints and managing user input requests. Because an ontology can be perceived as an interactive system where multiple viewpoints of the knowledge representation are required in order to complete reasoning tasks, it fits the application domain of the MVC-II architectural style. Moreover, when thinking about an ontology as a complete system that gives an understanding of a world, we can identify a clear division between the model, view, and controller components. Due to its modularity, the MVC-II architectural style can offer enhanced modifiability, extendability, and maintainability of the developed ontology. Additionally, it is effective for use in collaborative development environments which can better support reusability and shareability.

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

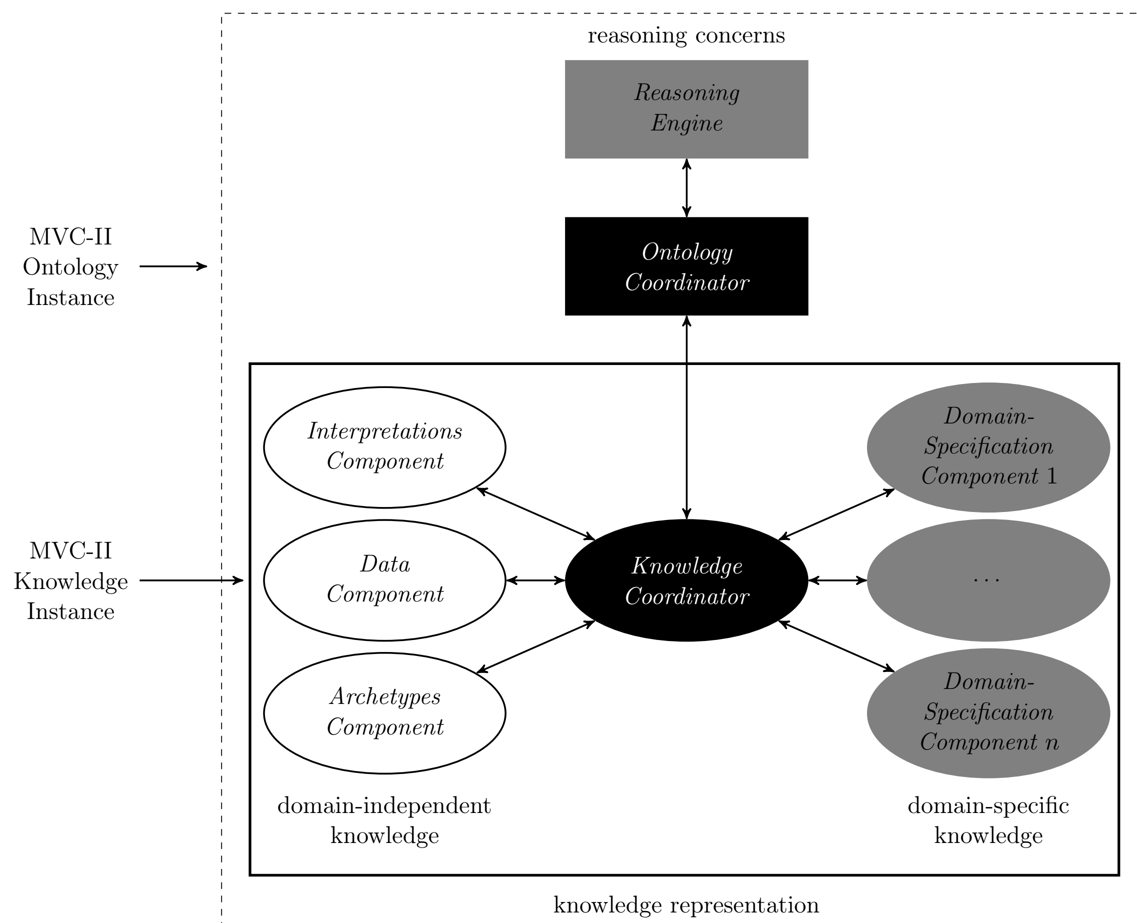

Figure 1: A nested MVC-II architecture for ontology development and design

3.1 An Example Ontology Design Scenario

Suppose that after performing a requirements and specification phase of ontology development similar to what is proposed by METHONTOLOGY, we are required to design an ontology that is capable of representing and reasoning on knowledge related to students, teachers, and the grades that have been achieved in the courses in which they are enrolled, or in which they teach, respectively. One of the ways in which we can outline the requirements of an ontology is to identify a list of questions that the knowledge contained within the ontology ought to answer. These questions are commonly called *competency questions* [23]. For the purpose of this example, assume that we are given the following competency questions:

1. Which student has the highest grade in Science 101?
2. Which teacher has the highest achieving students?
3. How many students have consistently maintained their grades each semester?

Using these requirements as a basis for illustration, we describe each of the components of the proposed MVC-II-based ontology design architecture in the following sections. Note that this simple example is purposely designed to highlight particular aspects and features of the proposed architecture.

3.2 The Knowledge Instance of the Proposed Architecture

The *Knowledge Instance* provides an MVC-II-based architecture for the knowledge representation of the ontology being developed. As such, it facilitates the separation of the domain-independent and domain-specific knowledge. The *Archetypes Component*, *Data Component*, and *Interpretations Component* constitute the *Knowledge Instance model* which encompasses the domain-independent knowledge representation of the ontology being developed. The collection of *Domain-Specification Components* constitutes the *Knowledge Instance view* which provides the domain-specific interpretations of the knowledge, thereby offering different viewpoints of the model component. Lastly, the *Knowledge Coordinator* constitutes the *Knowledge Instance controller* which manages the communication between the components representing the domain-independent and domain-specific knowledge in order to comprise a knowledge representation that models the worlds considered by the ontology.

3.2.1 Archetypes Component

The *Archetypes Component* is concerned with identifying a set of concepts and their respective attributes required to model the world dictated by the requirements of the ontology being developed. In this way, this component is related to identifying and developing *archetypes* that capture the conceptual knowledge contained within the ontology. It provides the general structural representation of the required concepts and their corresponding attributes in the form of a *conceptual skeleton*. This involves specifying the types of attributes that describe each concept. Such a *conceptual skeleton* is suitable for use or reuse in many domains as the archetypes for the required concepts are persistent in many possible worlds.

With respect to the design scenario outlined in Section 3.1, we identify the archetypes shown in Table 1 that have a role in modelling the world dictated by the requirements and competency questions. Each of these archetypes can be viewed as a module of the *Archetypes Component*. It should be noted that the level of granularity at which to identify the required archetypes is dependent on the requirements of the ontology being developed. We need to choose a granularity that appropriately captures the required knowledge in order to answer the given competency questions. For example, if we do not require a distinction between the *first name* and *last name* of a Person, then we can simply consider a PersonName as a String and we need not further articulate the concept PersonName with its own archetype capturing its finer-grained attributes.

3.2.2 Data Component

The *Data Component* is concerned with the data or information provided by the problem and requirements of the ontology being developed. It is responsible for providing a collection of facts by identifying instances (individuals) of the concepts and their attributes provided by the *Archetypes Component*. One can think of the *Data Component* as an interface to the data sources that are providing the facts for the given problem.

Assume that for the design scenario outlined in Section 3.1, we have a number of data sources resulting from a knowledge acquisition phase. The first collection of data represents student-filled data sheets of their own personal records of their grades. The second collection of data represents the grade sheets for the courses that each teacher teaches. A sample of each of these data sources is shown in Table 2 and Table 3, respectively.

Table 1: Concepts and attributes corresponding to the archetypes identified for inclusion in the *Archetypes Component*

Concept	**Attributes**
Person	*name* : PersonName; *id* : Integer;
PersonName	*first* : String; *last* : String;
Course	*name* : String; *code* : Integer;
Grade	*grade* : GradeType;
Semester	*season* : oneOf{Spring, Summer, Fall, Winter}; *year* : Integer; *coursesOffered* : ItemsType⟨Course⟩;

Table 2: Course and grade data provided by student John Smith

Student	**Semester**	**Course**		**Grade**	**Teacher**
John Smith	Fall 2014	English	101	A-	Hank Jones
		Math	101	B+	William Taylor
		Science	101	A-	Alice Brown
	Winter 2015	English	102	A	Bill Thompson
		Math	102	B-	Rhonda Wilson
		Science	102	A+	Alice Brown

Table 3: Course and grade data provided by teacher Alice Brown

Teacher	**Semester**	**Course**		**Student**	**Grade**
Alice Brown	Fall 2014	Science	101	John Smith	A-
				Sally McHugh	B
				Anne Rickman	C-
				Harvey Walker	B
				Scott Baker	F
	Winter 2015	Science	102	John Smith	A+
				Sally McHugh	B
				Anne Rickman	C+
				Harvey Walker	B-
				Stacey Nelson	A

The idea with the *Data Component* 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 we are able to instantiate the concepts defined in the *Archetypes Component* and the *Domain-Specification Components* with assertions about the domain of interest and the data that is collected through knowledge acquisition.

3.2.3 Interpretations Component

The *Interpretations Component* is concerned with providing concrete interpretations for the abstract types that are specified in the *Archetypes Component* and instantiated by the *Data Component*. It allows the ontology being developed to contain concepts and relationships that have different embodiments in different possible worlds. Moreover, it separates these possible interpretations of the concepts and relationships from the concepts and relationships themselves. The notion of concepts and relationships having different embodiments is not currently considered in the existing literature.

Consider the design scenario given in Section 3.1. Depending on the context of the competency questions and the requirements of the ontology being developed, the abstract GradeType identified in Table 1 can have different interpretations. For example, 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). Similarly, the design scenario suggests that a Semester offers a collection of courses. Again, depending on the context, this abstract collection type (denoted by ItemsType in Table 1) can be concretely interpreted as a *list*, *set*, *bag*, or *sequence*, for instance. There are plenty of other examples of potential abstract types that may be considered and that require interpretations to be included in the *Interpretations Component*. For instance, we can have a numeric type that can be concretely interpreted as a *real number*, *integer*, or *natural number*, or even something more exotic like an abstract type that allows a concept of Apple to be concretely interpreted as a *fruit*, *company*, or *computer model*.

The purpose of the *Interpretations Component* is to provide the definitions of the possible concrete interpretations of the abstract types that ought to be considered so that one (or more) may be selected for instantiation in the overall knowledge representation of the ontology. It is important to note that each of the possible interpretations defined in the *Interpretations Component* is independent of any particular domain and can be used in many different application domains. Furthermore, we conjecture that through the design of the *Interpretations Component*, the issue of *multiple inheritance* (e.g., [4, 12, 46]) can be avoided by allowing a concept to realise multiple interpretations and by deferring the choice over which interpretation to employ to later stages in the design and usage of the developed ontology, perhaps based on some user input.

3.2.4 Domain-Specification Components

The *Domain-Specification Components* are concerned with providing domain-specific knowledge in order to give specific viewpoints of the domain-independent knowledge contained within the *Archetypes Component*, *Data Component*, and *Interpretations Component*. Each viewpoint is encapsulated in a single *Domain-Specification Component* responsible for providing the particular interpretations of concepts and relationships within its specific viewpoint based on the requirements and application domain of the ontology being developed. For exam-

ple, given a data set related to information collected from a "smart city", different viewpoints such as transportation, public health, and education can be identified and encapsulated in individual *Domain-Specification Components* in order to focus on different concerns related to the data. Moreover, the *Domain-Specification Components* are concerned with identifying specialisations of the archetypal concepts identified in the *Archetypes Component*. In this way, a collection of *Domain-Specification Components* can allow for different viewpoints or domains of application to be considered while using the same *Archetypes Component*, *Data Component*, and *Interpretations Component*.

Concerning the design scenario from Section 3.1, we have a domain involving students, teachers, courses, and grades. We can develop two domain viewpoints, a student viewpoint and a teacher viewpoint, by considering the competency questions for the desired ontology. These viewpoints can be seen as specialisations of the Person archetype from the *Archetypes Component*. In this way, each of these viewpoints will inherit the attributes of the Person concept defined by the Person archetype and will extend it with domain-specific attributes. A similar extension is made for the Course archetype. These *Domain-Specification Components* are shown in Table 4 and Table 5, respectively. It can be seen from these domain-specific viewpoints that a number of domain-specific relationships are implicitly specified between the concepts identified in the *Archetypes Component* and *Domain-Specification Components*. For example, it is easy to see that a Student is *enrolled in* a collection of Courses, a Teacher *teaches* a collection of Courses, and a Student *has a grade* for each Course in which they are enrolled.

Table 4: Domain specification of the student viewpoint as specialisations of the Person and Course archetypes from the *Archetypes Component*

Concept	**Attributes**
Student	*student* : Person; *coursesEnrolled* : ItemsType⟨EnrolledCourse⟩;
EnrolledCourse	*course* : Course; *gradeEarned* : Grade;

Table 5: Domain specification of the teacher viewpoint as specialisations of the Person and Course archetypes from the *Archetypes Component*

Concept	**Attributes**
Teacher	*teacher* : Person; *coursesTaught* : ItemsType⟨TaughtCourse⟩;
TaughtCourse	*course* : Course; *students* : ItemsType⟨Student⟩;

3.2.5 Knowledge Coordinator

The *Knowledge Coordinator* is responsible for coordinating the domain-independent knowledge contained within the *Archetypes Component*, *Data Component*, and *Interpretations Component* and the domain-specific knowledge contained within the *Domain-Specification Components*. It 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*. In essence, this component is a controller that coordinates the interpretation of data from the *Data Component* as concepts from the *Archetypes Component*, possibly specialised by the domain-specific interpretations from the *Domain-Specification Components*, along with concrete type interpretations from the *Interpretations Component*.

With respect to the design scenario in Section 3.1, the functionality of the *Knowledge Coordinator* allows for the ability to state that *John Smith* is a Student *enrolled in* the Course *Science 101 taught by* the Teacher *Alice Brown* and in the *Fall 2014* Semester and has earned a Grade of *A-* which is interpreted as a *letter-grade*. Through this example, it is easy to see that the *Knowledge Coordinator* manages all of the domain-independent and domain-specific knowledge so that it can be used to capture a particular view of the world which can, in turn, be used for particular reasoning tasks in the developed ontology.

3.3 The Ontology Instance of the Proposed Architecture

The *Ontology Instance* provides an MVC-II-based architecture for the overall ontology being developed and provides separation of the knowledge representation and reasoning concerns. The *Knowledge Instance* (denoted by all of the elliptical components in Figure 1) constitutes the *Ontology Instance model* which provides the *knowledge representation* of the ontology. The *Reasoning Engine* constitutes the *Ontology Instance view* which provides an interface to existing reasoning tools and enables the consideration of various configurations of the knowledge representation with respect to different possible worlds in the form of different interpretations and/or viewpoints. Finally, the *Ontology Coordinator* constitutes the *Ontology Instance controller component* 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.

3.3.1 Reasoning Engine

The *Reasoning Engine* is concerned with the reasoning tasks that are required of the ontology being developed and is responsible for interfacing with existing reasoning tools. It allows for the specification of different reasoning approaches based on given theories, allowing the proposed architecture to handle complex reasoning tasks in a variety of ways. Since the *Reasoning Engine* represents a view component of the MVC-II architectural style, we can have many different *Reasoning Engine* components, each representing a different kind of reasoning or a different reasoning task. This allows us to use the same kind of reasoning to perform different reasoning tasks or similarly, different kinds of reasoning to perform the same reasoning task. As an example of one possibility, we may have one component responsible for association rule mining, and one component responsible for data cleansing, where each may use tableau-style reasoning, for instance. In this case, each component is responsible for performing a particular reasoning task based on a specified theory and using the same kind of reasoning. Consider the design scenario from Section 3.1. We can design a *Reasoning Engine* component to mine association rules from the data of students and teachers using a specified kind of reasoning and a given theory. In this way, this component is responsible for identifying relationships among the given set of values. The association rules that are identified can be then be used with another *Reasoning Engine* component designed to perform a data cleansing task. The use of association rule mining to facilitate data cleansing is common in the literature (e.g., [26]).

The specification of knowledge and information management approaches that should be considered in order to address inconsistent or conflicting information contained in the ontology are also defined in the *Reasoning Engine*. Once again, consider the design scenario given in Section 3.1. It is possible that in the *Data Component*, we have conflicting information from each of the given data sources. For instance, a teacher for a particular course may report a grade for a student, while that student may report a different grade that they achieved in the same course reported by the teacher. There are a number of existing approaches for handling such situations, including belief revision (e.g., [3, 13, 17]), among others (e.g., [8, 30, 31, 47]). Which to use and how to handle each case of conflicting information is ultimately dependent on the requirements and competency questions of the ontology being developed. For instance, we can choose to take the grade assigned and reported by the teacher to be the actual data, since it may be the case that we trust the teacher more than the student. In another case, we may mark the data with the source from which it came in order to keep track of both possibilities.

3.3.2 Ontology Coordinator

The *Ontology Coordinator* is concerned with handling the reasoning tasks from the *Reasoning Engine* and the knowledge representation of the ontology from the *Knowledge Instance* of the proposed architecture in order to ensure that the ontology being developed is fit for its purpose. In this way, it is responsible for selecting the appropriate domain-specific viewpoints, concrete interpretations, data sources, etc., that are required to perform the reasoning tasks from the *Reasoning Engine*. In simple terms, the *Ontology Coordinator* provides the bridge between the knowledge representation and the reasoning concerns of the developed ontology.

Concerning the design scenario from Section 3.1, consider the question: "Which student has the highest grade in Science 101?" For this example, suppose that the user indicates that it only wishes to consider the *set* interpretation of the collection of students enrolled in *Science 101* since it provides a sufficient level of expressivity in order to answer the question. Also, assume that the user only wishes to consider the data provided by the teacher of *Science 101*. In this case, the *Ontology Coordinator* is responsible for handling the user input passed from the *Reasoning Engine* in order to decide how to communicate the requirements of the reasoning task to the *Knowledge Coordinator*. This communication allows the student domain-specific viewpoint, and the *set* interpretation of the collection of students enrolled in *Science 101* to be selected and used to find the name of the Student with the maximum grade using the data provided by the Teacher of *Science 101*.

4 Merit of the Proposed Ontology Design Architecture

In this section, we discuss the benefits and drawbacks of the proposed MVC-II-based ontology design architecture.

4.1 Benefits of the Proposed Architecture

In the area of software engineering, *separation of concerns* is a principle that emphasises the separation of design decisions that are likely to change, thereby protecting the other parts of a system from extensive modification if a design decision is changed [20]. In general, adherence to the principle of separation of concerns leads to modular systems which yield benefits in terms of

the flexibility of the system, including the ability to handle modifications and extensions [38]. Furthermore, by expressing concerns in an architectural model, the qualities of interest are built into a framework that can later be mapped to in order to address different needs, while inheriting the characteristics of the architectural model [37]. For these reasons, we conjecture that a number of beneficial qualities inherited from the proposed architectural framework will be exhibited in the developed ontology. A dedicated user study is required in order to fully evaluate and assess the effectiveness of the proposed architecture and is left as future work.

The proposed architecture provides a separation of concerns at multiple levels. First, the *Ontology Instance* separates the knowledge representation and reasoning concerns of the ontology being developed. Second, the *Knowledge Instance* separates the domain-independent and domain-specific knowledge which enables 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. Because of the separation of concerns provided by the proposed architecture, the developed ontology will exhibit enhanced modifiability, extendability, and maintainability. A developer only needs to identify the concern of a required modification or extension in order to locate which component(s) require changes. For example, if a new domain-specific viewpoint of the knowledge is required for an existing ontology developed using the proposed architecture, then the developer needs simply to add a new *Domain-Specification Component* in the *Knowledge Instance view* component that can be registered with the *Knowledge Coordinator*. The rest of the components remain intact. This enables the developed ontology to be maintained over an extended period of time. Also, the developed ontology can benefit from enhanced reusability. For example, the *Archetypes Component*, *Data Component*, and *Interpretations Component* from the *Knowledge Instance model* component 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 of the developed ontology can be reused in order to address a number of different reasoning concerns and to answer different questions in the domain of interest. These are just some of the many ways in which the components of the proposed architecture can be reused in the development of new ontologies or in the modification and extension of existing ontologies. Further discussion, along with examples that illustrate the qualities exhibited by an ontology designed using the proposed architecture are provided in Section 5.3.

4.2 Drawbacks of the Proposed Architecture

While there exists a number of benefits resulting from the adoption of the proposed architecture, there are drawbacks related to the amount of communication overhead and the complexity of the controller components. The design of the *Knowledge Coordinator* and the *Ontology Coordinator* is a tricky problem. In general, the design of the controller components is not necessarily straightforward. Much effort is required in order to adequately handle the amount of communication necessary to facilitate the coordination of the other components in the system. This additional controller complexity is an inherent drawback of the use of the MVC-II architectural style [41]. Furthermore, much care needs to be taken in designing the controller components to ensure that the correct knowledge and information are available to, and communicated by, the *Knowledge Coordinator* and the *Ontology Coordinator* so that queries can be answered properly. A poorly designed controller component can effectively render the system unusable and is a problem that must be investigated. To address this issue,

a robust framework to support and facilitate the practical implementation of ontologies using the proposed architecture is needed. The development of such a framework is left as future work.

When it comes to engineering design, there is often a trade-off between the benefits of separation of concerns and the drawback of the amount of communication and complexity required to manage and coordinate the system components. Since the existing literature of ontology development finds so many issues with respect to the ability to easily reuse, extend, modify, and maintain ontologies, we conjecture that the proposed architecture makes an appropriate trade-off in this regard. It should be noted that while a significant amount of effort may be required in order to design and implement the controller components of the proposed architecture, it is the case that once these components are developed, they are relatively stable and are not prone to frequent changes, meaning that they can be reused for a long time afterwards.

5 Designing an Ontology Using the Proposed Architecture

In order to further illustrate the usage and benefits of the proposed architecture, consider redesigning the well-known *Wine Ontology* [1]. The *Wine Ontology* has been developed as a simple illustrative example of an ontology used for answering questions about particular wines, their characteristics, and the regions and wineries in which they are made. For the purpose of this section, suppose that we are given the requirements and competency questions for the *Wine Ontology* as a result of the requirements and specification phase of ontology development. In what follows, we outline how to design the *Wine Ontology* using the proposed architectural framework. For simplicity and brevity, we will keep the design of the *Wine Ontology* at a coarse granularity in order to highlight the benefits of designing ontologies using the proposed architecture.

5.1 Designing the Knowledge Instance of the Proposed Architecture

We begin the process of designing the *Wine Ontology* by designing the inner *Knowledge Instance* of the nested MVC-II architecture.

5.1.1 Designing the Archetypes, Data, and Interpretations Components

We start by designing the *Archetypes Component*. We identify a number of archetypes that play a role in modelling the world in which we are interested in capturing. The identification is based on the requirements and competency questions for the *Wine Ontology*. The archetypes are shown in Table 6.

Next, we design the *Data Component* as an interface to the possible data sources that may be available for the *Wine Ontology*. For the purpose of the example, suppose that we simply have a database containing various wine data obtained through the knowledge acquisition phase of ontology development. A fragment of such a database may resemble what is given in Table 7.

Then, with consideration to the *Interpretations Component* and the *Wine Ontology*, it is possible that we may need to consider different interpretations of collection of wines that are made by a particular winery, or the collection of wineries that exist in a particular region. For instance, the collection of wines that are made by a particular winery can be interpreted as

Table 6: Concepts and attributes corresponding to the archetypes identified for inclusion in the *Archetypes Component* for the *Wine Ontology*

Concept	**Attributes**
Wine	*body* : someOf{Full, Medium, Light}; *colour* : oneOf{Red, Rose, White}; *flavour* : someOf{Strong, Moderate, Delicate}; *sugar* : oneOf{Dry, OffDry, Sweet}; *grape* : Grape;
Grape	*name* : String;
Region	*name* : String;
Winery	*name* : String; *winesMade* : ItemsType⟨Wine⟩;

Table 7: A fragment of wine data for the *Data Component* of the *Wine Ontology*

Wine	**Body**	**Colour**	**Flavour**	**Sugar**	**Grape**	**Region**
Zinfandel	Full, Medium	Red	Moderate, Strong	Dry	Zinfandel	—
Sweet Riesling	Full	White	Moderate, Strong	Sweet	Riesling	—
St. Emilion	—	Red	Strong	Dry	Cabernet Sauvignon	French
⋮	⋮	⋮	⋮	⋮	⋮	⋮

a *set*, *list*, or *bag*, depending on the context in which this knowledge needs to be used. For example, in a context where the collection of wines made by a particular winery needs to be ordered, then the *list* interpretation is appropriate as it allows for the order of the items to be captured. Similarly, in a context where we are interested in only the unique wines made by a particular winery, then the *set* interpretation is appropriate as it will remove any duplicate wines from the collection. Therefore, the *Interpretations Component* for the *Wine Ontology* must contain these possible interpretations so that they are available for selection by the *Knowledge Coordinator* for use in addressing different reasoning concerns based on different contexts. Note that current versions of the *Wine Ontology* found in the literature do not consider this notion of concepts and relationships having different interpretations in different possible worlds and contexts.

5.1.2 Designing the Domain-Specification Components

With respect to the *Wine Ontology*, there can be a number of different *Domain-Specification Components*. The selection of the *Domain-Specification Components* that need to be designed depend on the requirements, competency questions, and overall purpose of the ontology. For brevity, suppose that we are interested in providing three domain-specific viewpoints for the *Wine Ontology*. The first view pertains to the French regions in which a wine may be made, the second view pertains to the American regions in which a wine may be made, and the

third view pertains to the definition of table wines. The first two of these viewpoints can be seen as specialisations of the Region archetype, and the third viewpoint can be seen as a specialisation of the Wine archetype. The *Domain-Specification Components* corresponding to each of these viewpoints are shown in Table 8, Table 9, and Table 10, respectively. From these domain-specific viewpoints, a number of domain-specific relationships are implicitly being specified between each of the concepts identified in the *Archetypes Component* and in the *Domain-Specification Components*. For example, it is easy to see that a Winery is *located in* a particular Region, and that a TableWine *has sugar* corresponding to Dry.

Table 8: Domain specification of the French region viewpoint

Concept	**Attributes**
FrenchRegion	*region* : Region; *region.name* := "French"; *wineries* : ItemsType⟨Winery⟩;

Table 9: Domain specification of the American region viewpoint

Concept	**Attributes**
USRegion	*region* : Region; *region.name* := "US"; *wineries* : ItemsType⟨Winery⟩;

Table 10: Domain specification of the table wine viewpoint

Concept	**Attributes**
TableWine	*wine* : Wine; *wine.sugar* := Dry;

5.1.3 Guidelines for Designing the Knowledge Coordinator

With respect to the *Wine Ontology*, the *Knowledge Coordinator* is responsible for coordinating the *Knowledge Instance* components in order to instantiate concepts with facts and domain-specific relationships. For instance, the functionality of the *Knowledge Coordinator* must be designed to allow for statements to be constructed from the knowledge contained in the *Knowledge Instance* components. For example, with a properly designed *Knowledge Coordinator*, we ought to be able to say that *St. Emilion* is a TableWine *located in* the FrenchRegion. Through this example, it is easy to see that the *Knowledge Coordinator* manages the domain-independent and domain-specific knowledge to be used for particular reasoning tasks.

5.2 Designing the Ontology Instance of the Proposed Architecture

The process of designing the *Wine Ontology* continues with the design of the outer *Ontology Instance* of the nested MVC-II architecture.

5.2.1 Guidelines for Designing the Reasoning Engine

Suppose that we have a reasoning task related to data cleansing for the *Wine Ontology*. The goal of this reasoning task is to identify any contaminated data entries in the data contained within the knowledge representation of the ontology (i.e., the *Knowledge Instance*). For the purpose of this example, suppose that we have the following fragment of a theory about wines[2]:

- All *Zinfandel* wines have a *Moderate* or *Strong* flavour.
- If a wine is *Strong*, *Red*, and *Dry*, then it is made with a *Cabernet Sauvignon* grape and it is a *Table Wine*.
- All *Sweet Riesling* wines are *White* and are not *Table Wines*.

This theory is specified (using any suitable formalism) within the *Reasoning Engine* component and is used to identify any contaminated data entries. For instance, if we have a data entry that stated that a *Zinfandel* wine had a *Delicate* flavour, then with respect to the given theory, the data cleansing reasoning task would identify this entry as being contaminated.

The current form of the *Wine Ontology* used in this example does not contain any threat of conflicting information from the data sources in the *Data Component*. Therefore, the *Reasoning Engine* does not require any specification of knowledge and information management approaches. However, it should be noted that such approaches could easily be incorporated if and when they are required. For this example, the *Reasoning Engine* can provide hints based on the user input to aid reasoners in obtaining answers to the questions that the ontology is designed to answer. For instance, to answer the question "What are the three most popular wineries in the American region?", we can specify that the knowledge representation should be configured to indicate that we only wish to consider the world in which the collection of wineries in the American region is interpreted as a *list* since the question indicates that it is necessary to order the wineries according to some pre-defined criteria (which are not currently represented in this example). In this way, the reasoner does not need to consider any of the other possible interpretations of this collection. We conjecture that these kinds of hints based on the user input can offer savings in terms of the time and computation power required to answer the query by removing the need to consider worlds that are not deemed possible in the current context.

5.2.2 Guidelines for Designing the Ontology Coordinator

With respect to the *Wine Ontology*, the *Ontology Coordinator* is responsible for coordinating the reasoning tasks from the *Reasoning Engine* and the knowledge representation of the ontology from the *Knowledge Instance* of the proposed architecture. For example, consider using the *Wine Ontology* to answer the question "How many table wines are produced in the French region?" The *Ontology Coordinator* is responsible for managing the user input and configurations specified in the *Reasoning Engine* and for communicating with the *Knowledge Coordinator*. Through this communication, the table wine and French region domain-specific viewpoints and the *set* interpretation of the collection of wineries can be selected in order to reason on and determine the number of wines satisfying the criteria in order to obtain an answer to the question. This is an example where more than one domain-specific viewpoint is required to answer a query.

[2]The authors apologise to those readers that are wine connoisseurs for their ignorance to the factual correctness of the statements captured in the proposed theory. It is meant strictly for illustrative purposes.

5.3 Discussion

It is important to emphasise that the proposed architecture and its corresponding design processes do not interfere with the existing tasks in current ontology development methodologies such as METHONTOLOGY. It is meant to complement the current conceptualisation phase in order to provide additional structure in the form of an architectural design framework. By adopting this design architecture, the resulting ontology can exhibit a number of desirable qualities as mentioned in Section 4. We discuss the benefits of designing an ontology using the proposed architecture by using the design of the *Wine Ontology* developed above.

5.3.1 Support for Modifiability

Suppose that the definition of a table wine needs to be updated so that a table wine is not only dry, but that it can only be red. In order to make this modification, we simply need to identify that the proposed modification needs to occur in the *Domain-Specification Component* pertaining to table wines. Specifically, we modify the domain-specific details shown in Table 10 to provide a modified table wine *Domain-Specification Component* as shown in Table 11 where the modification is highlighted in boldface.

Table 11: Domain specification of the modified table wine viewpoint

Concept	Attributes
TableWine	*wine* : Wine; *wine.sugar* := Dry; ***wine.colour*** **:= Red;**

Consider making this modification in the *Wine Ontology* found in the literature. Because there is no clear separation of concerns, it can be quite difficult to locate where the modification needs to be made. Furthermore, it can be even more difficult to ensure that the modification is complete since it is possible that the modification needs to be made in more than one area of the ontology. The lack of guidelines for modifying existing ontologies is noted as an issue in the current ontology development literature [15]. However, with the proposed architecture, the identification of the component(s) in which a modification needs to be made is rather straightforward and proceeds in a systematic way by examining the constituent components of the ontology design. Moreover, when performing this modification, all of the other components in the design of the *Wine Ontology* remain intact and do not require any additional modifications.

5.3.2 Support for Extendability

Suppose that after we have developed our design of the *Wine Ontology*, new competency questions arise that must be handled. For the sake of illustration, suppose that the *Wine Ontology* also needs to address the following questions:

1. How many types of grapes grow in a Canadian region?
2. What is the chemical composition of a Table Wine?

The introduction of these new competency questions indicate a change in the requirements of the ontology being developed. To satisfy these new requirements, we need to iterate the design process and extend our current design.

In order to address the first of the newly introduced competency questions, we need to represent and reason on knowledge related to the kinds of grapes that grow in a Canadian region. This can be done with the addition of a new domain-specific viewpoint that captures this requirement. This new *Domain-Specification Component* is shown in Table 12. In this case, this new component simply needs to be included and registered with the *Knowledge Coordinator* in order to be used for the required reasoning tasks. Once again, all of the other components in our design remain intact. However, considering the *Wine Ontology* found in the literature, it may be difficult to determine where to make the required updates and changes in order to perform the same extension in a straightforward manner.

Table 12: Domain specification of a Canadian region viewpoint concerned with the kinds of grapes that grow in the region

Concept	**Attributes**
CanadianRegion	*region* : Region; *region.name* := "Canadian"; *grapesGrown* : ItemsType⟨Grape⟩;

The introduction of the second competency question involves more changes. It requires the introduction of new archetypes, data, and interpretations, each pertaining to chemical compositions of wines. Since every wine has a chemical composition, we first extend the *Archetypes Component* with archetypes pertaining to the general concepts and attributes related to the chemical composition of wines. The extended *Archetypes Component* is shown in Table 13 where the newly added concepts and attributes are highlighted in boldface. In this case, we add a *chemicals* attribute to a Wine which indicates the collection of chemical compounds and the percentage of each that form the chemical composition of the wine. For simplicity, we consider the concept of Chemical to represent a chemical compound. Once again, depending on the specific context in which this concept needs to be used, a much finer granularity can be articulated. Additionally, we consider a Percentage as a concept with an abstract numeric type.

Continuing with the required extension, we need to update the *Data Component* in order to include data related to the chemical composition of particular wines. For example, such data may indicate that a *St. Emilion* wine has the following chemical composition: {(*water*, 86.0), (*ethanol*, 12.0), (*glycerol*, 1.0), (*organic acids*, 0.4), (*tannins*, 0.1), (*other*, 0.5)}. Similarly, the *Interpretations Component* needs to be extended to include the additional interpretations that are required in the extended version of the *Wine Ontology*. For example, the changes made in the *Archetypes Component* introduce a new abstract NumericType. In this case, the numeric type represents a percentage and can be concretely interpreted as a *real number* (e.g., 15.0%), an *integer* (e.g., 15%), or a *rational number* (e.g., 3/20). The *Interpretations Component* needs to be updated to reflect these possible interpretations of the composition percentage of the chemicals in a wine.

As shown by the above example, extensions to ontologies can have crosscutting concerns that require changes to be made in different components of the ontology. Due to the separation

Table 13: Extended *Archetypes Component* for the *Wine Ontology*

Concept	Attributes
Wine	*body* : someOf{Full, Medium, Light}; *colour* : oneOf{Red, Rose, White}; *flavour* : someOf{Strong, Moderate, Delicate}; *sugar* : oneOf{Dry, OffDry, Sweet}; *grape* : Grape; ***chemicals* : ItemsType⟨(Chemical, Percentage)⟩;**
Grape	*name* : String;
Region	*name* : String;
Winery	*name* : String; *winesMade* : ItemsType⟨Wine⟩;
Chemical	***name* : String;**
Percentage	***num* : NumericType;**

of concerns offered by the proposed architecture, we have a systematic way of performing the required extension. The architectural design of the ontology provides a guide that enables the consideration of each concern, one at a time, by examining the changes that are required in each component. This is in contrast to how such an extension would proceed in the *Wine Ontology* found in the literature. Because of the crosscutting nature of the required extension, in the *Wine Ontology* found in the literature, it is difficult to be sure which areas of the ontology are affected by the extension and if the extension is even complete, resulting from the absence of guidelines for extending and maintaining existing ontologies [15].

5.3.3 Support for Reusability

Consider the *Wine Ontology* found in the literature. It is not clear which parts can be reused in the development of other ontologies or how they can be reused. In current approaches for developing ontologies found in the literature, the search for parts of ontologies that can be reused is difficult, time-consuming, and often fruitless [49]. However, consider our design of the *Wine Ontology*. A number of the components that exist in the design can be reused, either as is, or with moderate modifications in other ontology designs. For example, much of the domain-independent knowledge contained in the *Archetypes Component*, such as the Region archetype and Wine archetype — although relatively simple in the given example — may be reused in other ontologies that are concerned, for instance, with geography or food, respectively. Similarly, the *Data Component* can be reused with different data sources "plugged-in" to offer different sets of facts for another ontology design. The same can be said for the *Knowledge Coordinator* and *Ontology Coordinator* as these components can be largely reused with moderate modifications in other ontology designs.

5.3.4 Support for Collaborative/Parallel Development

Due to the separation of concerns and the resulting modularity of the proposed architecture, it is possible to develop an ontology in a collaborative environment. This enables a number of different components to be developed in parallel provided that a kind of interface (i.e., the con-

cepts and attributes that are registered with the controller components) is agreed upon among the different development teams before development. This sort of interface can often be fleshed out of the existing conceptualisation phase of existing ontology development methodologies like METHONTOLOGY. With respect to the *Wine Ontology*, this is particularly apparent when considering the design and development of the *Domain-Specification Components*. To be clear, we do not claim that every component of the ontology design can be developed in parallel with one another, since it is the case that the design of some components depends on the design of others. However, we conjecture that with parallel and collaborative development for some components, when compared to sequential development, the development time for an ontology can be reduced (e.g., [9, 32]). Furthermore, the reusability and shareability of the developed ontology and its components can be improved.

5.3.5 Support for Concurrent and Distributed Reasoning

For simplicity of presentation, we discussed an MVC-II-based ontology design architecture for use in cases where concurrent and distributed reasoning is not required. However, in order to handle cases where these kinds of reasoning are needed, such as those faced by "big data" applications, the proposed architecture can be extended to a presentation-abstraction-control (PAC) architectural style. The PAC architectural style was developed as an extension of MVC-II to support multiple agents each with their own functionalities with data and interactive interfaces. It is best suited for interactive systems that can be divided into many cooperating agents in a hierarchical structure [41].

To illustrate the idea of extending the proposed architecture to a PAC architectural style supporting concurrent and distributed reasoning, let us consider a simple example. Suppose that when designing the *Wine Ontology*, we are faced with a "big data" scenario where we have an enormous amount of information corresponding to wines, regions, wineries, chemical compositions, etc.[3] Once again, consider a reasoning task related to data cleansing using the *Wine Ontology* where the goal is to identify any contaminated entries in the data set with respect to a given theory (see Section 5.2.1 for an illustrative theory that can be used for data cleansing in the *Wine Ontology*). In order to support concurrent and distributed reasoning in this case, we can divide the problem into multiple fragments and employ a PAC architectural style. By designing different agents to work with different problem fragments, each agent can accomplish the data cleansing task using its own specified theory and viewpoint. For instance, we can have one agent that approaches the problem from the viewpoint of the chemical composition of wines. In this case, this agent can be designed with a given theory about the chemical composition of wines that it can use for a data cleansing task. Other agents can be given a similar task with respect to a region viewpoint, or a grape viewpoint, each with its own given theory. In this way, each agent works with the entire data set to identify contaminated entries with respect to their given theory and particular viewpoint. This is in contrast to having a single agent with a large, monolithic, and complicated theory. Since each agent is designed using the proposed MVC-II-based architecture, the *Ontology Coordinator* of each agent represents the agent controller for the PAC architectural style. Higher-level agents can then be designed and arranged in a hierarchy in order to coordinate each agent to work on their designated tasks. In this way, for the data cleansing reasoning task, each agent can

[3]The authors would like to remind the reader that while this example does not define "big data" in the traditional sense, it is meant for the sake of illustration and to demonstrate the scalability the proposed architecture.

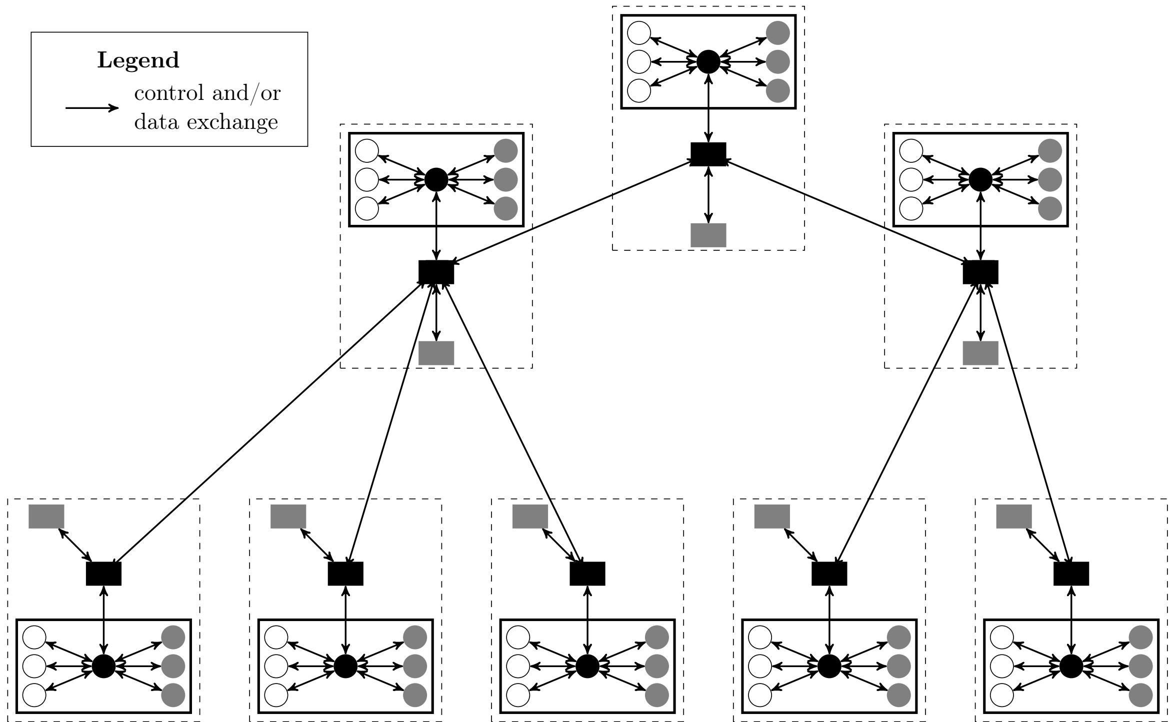

Figure 2: An illustration of extending the proposed MVC-II architecture to a PAC architecture to support concurrent and distributed reasoning

work independently on their fragment of the problem and report its results to the high-level agents which will work together to combine the results to give a final answer. This idea can be extended further by considering agents that use different reasoning approaches or that have different ways in which they handle conflicting information in the given data set.

It is important to note that it is not necessary that each agent in the PAC architecture does the same job. For instance, we may have one agent responsible for data cleansing, and another agent responsible for data mining. However, another way to use the PAC architectural style to deal with "big data" applications is to employ it in a fashion similar to that of a Master-Slave architectural style. In this case, the data set is divided into separate fragments based on different dimensions and where each agent does the same job on its given fragment. Suppose that we divide the data set into fragments based on regions. For the purpose of illustration, assume that we have five data fragments denoted as *French*, *American*, *Canadian*, *Australian*, and *German*. To employ a PAC architectural style for this scenario, we can design an agent using the proposed MVC-II-based architecture to work with each data fragment. For our example, we will have one agent responsible for working with each of the five aforementioned data fragments. Here, each agent performs the same job, working independently of each other on their fragment of data to identify contaminated entries. Once the tasks are completed, each agent will report its results to be combined into a final answer by the higher-level agents.

The general idea of extending the proposed MVC-II architectural style to a PAC architectural style is shown in Figure 2 where the arrows represent the exchange of control and/or data. Each agent of the hierarchy represents a copy of the proposed MVC-II architecture (see Figure 1) responsible for working with each fragment of the problem according to its particular

viewpoint. The hierarchy of controller components for the PAC architectural can be layered as deeply as needed. A deeper hierarchy can allow for smaller tasks to be performed or a smaller data set to be handled. In this way, we can have a divide-and-conquer approach, either in terms of the task to be performed or the data set with which we need to work, that can help deal with "big data" scenarios.

The support for concurrent and distributed reasoning comes at the cost of development complexity. In fact, we have two kinds of complexity that play a role when considering concurrent and distributed reasoning. The first is related to the overall size of the data that needs to be considered in the reasoning. There is a trade-off between communication complexity and the size of data or problem fragments that are considered. The higher the number of fragments, the more communication complexity that is introduced in order to coordinate all of the PAC agents and to combine and return the results of the reasoning tasks. This is a problem that needs to be considered and evaluated before dividing the task or fragmenting the data sets when designing the ontology and its reasoners. The second is related to the inherent complexity of the theory with which we need to work in order to complete the given reasoning tasks. This becomes even more difficult to manage when the theory is divided among components and when dependencies among each part of the theory need to be dealt with. This problem is a consequence of the current issues and difficulties in dealing with modularisation in ontology design. Managing all of this complexity is not an easy problem to solve and the balance of the communication among the PAC agents with respect to the load, both in terms of size and theoretical complexity, for a given reasoning task needs to be addressed and examined further. Despite this additional complexity, we conjecture that extending the proposed architecture to support concurrent and distributed reasoning allows us to alleviate some of the burden of reasoning on very large data sets. This is a result of the fact that the effort for designing the controller components needs to be done only one time for each individual domain of interest and can then be reused for tasks within the same domain afterwards.

6 Conclusion and Future Work

Motivated by the lack of design consideration in ontology development and the need to represent and reason on vast amounts of data, we proposed an ontology design architecture. The proposed architecture adopts a nested MVC-II-based architectural style inspired by the software engineering field. It supports the principle of separation of concerns with respect to the knowledge representation and reasoning abilities of the developed ontologies, as well as with respect to the domain-independent and domain-specific knowledge required of an ontology to capture particular viewpoints of the possible worlds that it needs to consider. As a result, ontologies designed using the proposed architecture exhibit a number of desirable qualities, such as enhanced modifiability, extendibility, and reusability. They also have the potential to support collaborative/parallel development, and concurrent and distributed reasoning as discussed and illustrated in Section 5.3. These benefits address aspects of a number of the criticisms found in the literature with respect to the current state-of-the-art for ontology development, and the proposed architecture helps to eliminate the current ad hoc "one-time use" mentality of ontology development.

Currently, the proposed architecture outlines a framework that can be adopted in order to systemically design ontologies in a structured way. This leads to a more refined engi-

neering approach to ontology development. However, the details of practically adopting and using the proposed architecture as a framework to design real-world ontologies requires further investigation. A user study is required in order to evaluate and assess the effectiveness of the proposed architecture and its impact on ontology development. Furthermore, the use of the PAC architectural style to design ontologies with support for concurrent and distributed reasoning to address "big data" concerns needs to be investigated further. Also, the details of practically designing and implementing the controller components of the proposed architecture, namely the *Knowledge Coordinator* and *Ontology Coordinator*, require much more attention. The development of a robust framework capable of supporting and facilitating the practical implementation of ontologies using the proposed architecture is needed. This is the basis of our current and future work. We aim to create an infrastructure that shows how to practically use the proposed architecture, and we intend to explore how existing technologies can be employed in order to realise this goal.

References

[1] Wine ontology. Available: `http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine` (Accessed July 1, 2015).

[2] P. Adjiman, P. Chatalic, F. Goasdoué, M.-C. Rousset, and L. Simon. Distributed reasoning in a peer-to-peer setting: Application to the semantic web. *Journal of Artificial Intelligence Research*, 25:269–314, 2006.

[3] C. E. Alchourrón, P. Gärdenfors, and D. Makinson. On the logic of theory change: Partial meet contraction and revision functions. *The Journal of Symbolic Logic*, 50(2):510–530, 1985.

[4] M. E. Aranguren. Automatic maintenance of multiple inheritance ontologies. Available: `http://ontogenesis.knowledgeblog.org/49` (Accessed July 16, 2015), January 2010.

[5] T. Beale. Archetypes: Constraint-based domain models for future-proof information systems. In K. Baclawski and H. Kilov, editors, *Proceedings of the 11th OOPSLA Workshop on Behavioral Semantics: Serving the Customer*, pages 16–32, 2002.

[6] A. Bernaras, I. Laresgoiti, and J. M. Corera. Building and reusing ontologies for electrical network applications. In *Proceedings of the 12th European Conference on Artificial Intelligence*, ECAI '96, pages 298–302, 1996.

[7] E. Blomqvist. Ontology patterns – typology and experiences from design pattern development. In *Linköping Electronic Conference Proceedings*, pages 55–64. Linköping University Electronic Press, 2010.

[8] D. Calvanese, E. Kharlamov, W. Nutt, and D. Zheleznyakov. Evolution of DL-Lite knowledge bases. In *Proceedings of the 9th International Semantic Web Conference on The Semantic Web - Volume Part I*, ISWC'10, pages 112–128, 2010.

[9] A. K. Chakravarty. *Market Driven Enterprise: Product Development, Supply Chains, and Manufacturing*. Engineering and Technology Management Series. Wiley, 2001.

[10] 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, March 2014.

[11] O. Corcho, M. Fernández-López, and A. Gómez-Pérez. Methodologies, tools and languages for building ontologies. where is their meeting point? *Data & Knowledge Engineering*, 46(1):41–64, 2003.

[12] O. Corcho and A. Gómez-Pérez. A roadmap to ontology specification languages. In R. Dieng and O. Corby, editors, *Proceedings of the 12th International Conference on Knowledge Engineering and Knowledge Management: Methods, Models, and Tools*, volume 1937 of *Lecture Notes in Computer Science*, pages 80–96. Springer Berlin/Heidelberg, 2000.

[13] A. Darwiche and J. Pearl. On the logic of iterated belief revision. *Artificial Intelligence*, 89(1-2):1–29, January 1997.

[14] V. Devedžić. Understanding ontological engineering. *Communications of the ACM*, 45(4):136–144, April 2002.

[15] M. Fernández, A. Gómez-Pérez, and N. Juristo. METHONTOLOGY: From ontological art towards ontological engineering. In *Proceedings of the AAAI Spring Symposium on Ontological Engineering*, pages 33–40, Menlo Park, CA, USA, 1997. AAAI Press.

[16] M. Fernández-López, A. Gómez-Pérez, J. P. Sierra, and A. P. Sierra. Building a chemical ontology using methontology and the ontology design environment. *Intelligent Systems and their Applications*, 14(1):37–46, January 1999.

[17] N. Friedman and J. Y. Halpern. A knowledge-based framework for belief change. Part II: Revision and update. In *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning*, KR '94, pages 190–201, 1994.

[18] A. Gangemi and V. Presutti. *Handbook on Ontologies*, chapter Ontology Design Patterns. Springer Publishing Company, Incorporated, second edition, 2009.

[19] J. Geller, Y. Perl, and J. Lee. Editorial: Ontology challenges: A thumbnail historical perspective. *Knowledge and Information Systems*, 6(4):375–379, 2004.

[20] C. Ghezzi, M. Jazayeri, and D. Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, second edition, 2003.

[21] A. Gómez-Pérez. Ontological engineering: A state of the art. *Expert Update*, 2(3):33–43, 1999.

[22] A. Gómez-Pérez, M. Fernández-López, and O. Corcho. *Ontological Engineering: With Examples from the Areas of Knowledge Management, e-Commerce and the Semantic Web*, chapter 3: Methodologies and Methods for Building Ontologies, pages 107–197. Advanced Information and Knowledge Processing. Springer-Verlag New York, Inc., Secaucus, NJ, USA, first edition, 2004.

[23] M. Grüninger and M. S. Fox. Methodology for the design and evaluation of ontologies. In *Proceedings of the IJCAI-95 Workshop on Basic Ontological Issues in Knowledge Sharing*, 1995.

[24] K. Hammar. Ontology design patterns in use: Lessons learnt from an ontology engineering case. In *Proceedings of the 3rd Workshop on Ontology Patterns*, volume 929. CEUR Workshop Proceedings, 2012.

[25] K. Hammar. Ontology design patterns: Adoption challenges and solutions. In *Joint Proceedings of the Second International Workshop on Semantic Web Enterprise Adoption and Best Practice and Second International Workshop on Finance and Economics on the Semantic Web*, volume 1240. CEUR Workshop Proceedings, 2014.

[26] R. Khedri, F. Chiang, and K. E. Sabri. An algebraic approach towards data cleaning. *Procedia Computer Science*, 21:50–59, 2013. The 4th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2013) and the 3rd International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH).

[27] M. Krötzsch, P. Hitzler, M. Ehrig, and Y. Sure. Category theory in ontology research: Concrete gain from an abstract approach. Technical Report 893, Institute AIFB, Universität Karlsruhe, March 2005.

[28] D. B. Lenat and R. V. Guha. *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project*. Addison-Wesley, Boston, 1990.

[29] L. Lezcano, L. Santos, and E. García-Barriocanal. Semantic integration of sensor data and disaster management systems: The emergency archetype approach. *International Journal of Distributed Sensor Networks*, 2013:1–11, 2013.

[30] J. Lobo and G. Trajcevski. Minimal and consistent evolution of knowledge bases. *Journal of Applied Non-Classical Logics*, 7(1-2):117–146, 1997.

[31] J. Luo, Z. Shi, M. Wang, and H. Huang. Multi-agent cooperation: A description logic view. In D. Lukose and Z. Shi, editors, *Multi-Agent Systems for Society*, volume 4078 of *Lecture Notes in Computer Science*, pages 365–379. Springer Berlin/Heidelberg, 2009.

[32] P. Manickam, S. Sangeetha, and S. V. Subrahmanya. *Component-Oriented Development and Assembly: Paradigm, Principles, and Practice using Java*. Infosys Press. CRC Press, 2013.

[33] 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*, 43(5):736–746, 2010.

[34] OntologyDesignPatterns.org. Ontology design patterns, May 2014.

[35] openEHR Foundation. Archetype definitions and principles. Available: `http://www.openehr.org/releases/1.0.2/architecture/am/archetype_principles.pdf` (Accessed: July 16, 2015), March 2007.

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

[37] A. D. Pace and M. Campo. An empirical study about separation of concerns approaches. In *Proceedings of the 2nd Argentine Symposium on Software Engineering*, 2001.

[38] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972.

[39] H. S. Pinto and J. P. Martins. Ontologies: How can they be built? *Knowledge and Information Systems*, 6(4):441–464, 2004.

[40] V. Presutti and A. Gangemi. Content ontology design patterns as practical building blocks for web ontologies. In *Proceedings of the 27th International Conference on Conceptual Modeling*, pages 128–141, Berlin, Heidelberg, 2008. Springer-Verlag.

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

[42] G. Santipantakis and G. A. Vouros. Distributed reasoning with coupled ontologies: The E-$\mathcal{SHIQ}$ representation framework. *Knowledge and Information Systems*, pages 1–44, November 2014.

[43] F. Scharffe, J. Euzenat, and D. Fensel. Towards design patterns for ontology alignment. In *Proceedings of the 2008 ACM Symposium on Applied Computing*, SAC '08, pages 2321–2325, New York, NY, USA, 2008. ACM.

[44] Y. Sure and R. Studer. On-to-knowledge methodology (expanded version). EU-IST Project IST-1999-10132, University of Karlsruhe, April 2002.

[45] B. Swartout, R. Patil, K. Knight, and T. Russ. Towards distributed use of large-scale ontologies. In *AAAI Spring Symposium on Ontological Engineering*, 1997.

[46] V. A. Tamma and T. J. Bench-Capon. Supporting inheritance mechanisms in ontology representation. In R. Dieng and O. Corby, editors, *Proceedings of the 12th International Conference on Knowledge Engineering and Knowledge Management: Methods, Models, and ToolsKnowledge Engineering and Knowledge Management Methods, Models, and Tools*, volume 1937 of *Lecture Notes in Computer Science*, pages 140–155. Springer Berlin/Heidelberg, 2000.

[47] E. Teniente and A. Olivé. Updating knowledge bases while maintaining their consistency. *The VLDB Journal*, 4(2):193–241, April 1995.

[48] M. Uschold and M. King. Towards a methodology for building ontologies. In *Proceedings of the IJCAI-95 Workshop on Basic Ontological Issues in Knowledge Sharing*, 1995.

[49] J. C. A. Vega, A. Gómez-Pérez, A. L. Tello, and H. S. A. N. P. Pinto. (ONTO)^{2}Agent: An ontology-based WWW broker to select ontologies. In *Proceedings of 14th European Conference on Artificial Intelligence Workshop on Applications of Ontologies and Problem-Solving Methods*, ECAI-98, pages 16–24, 1998.

[50] D. A. Waterman. *A Guide to Expert Systems.* Teknowledge Series in Knowledge Engineering. Addison-Wesley, 1986.