Materializing deductions inferred by reasoning on a conceptual data model

Mandisa Baleni

Computer Science
University of Cape Town
blnman002@myuct.ac.za

ABSTRACT

The Semantic Web among other platforms are looking to represent knowledge in an ontology or conceptual data model in a way that the knowledge represented is expressive about the subject domain of which it interprets. Conceptual models need tools and methods to allow domain experts to encompass the features of the universe of discourse [1]. There are many benefits to having an expressive ontology conveys knowledge about a subject domain. Users can query data efficiently, knowing what they can query instead of taking on the task to discover the data of the domain and how its stored among other things. We need an intelligent system to properly capture such knowledge in such a way that all the implications of the subject domain are made explicit in the resultant ontology or conceptual data model. There have been several attempts at implementations of this yet there are no fully-automated systems able to do this effectively where knowledge is represented with full expressiveness.

CCS CONCEPTS

• Ontology • Conceptual data model• Extended Entity-Relationship model → EER

KEYWORDS

reasoner, formalisation, materialization of deductions

1 Introduction

In order to provide a better way of informed decision making, information systems need techniques for conceptual data modelling and handling ontologies [2]. New theories and techniques as well as tools have emerged to put forward using conceptual models at runtime with data storage [2] as a part of a larger system for the purpose of providing means to users making requests or queries about data. This makes it easier for a user to make queries about data because it reduces the task of having to find out what data exists and how it is stored and instead shows what they can query [2]. Some architectures of intelligent systems that provide the functionality of querying in such a way have been proposed. These include Ontology-Based Data Access and Knowledge-driven Information and Data access (KnowID) [2]. Such architectures include a component that manages knowledge where a given ontology or conceptual data model must be formalised, then a process of inferring implicit knowledge must take place and lastly those inferences must then be appended to the ontology or conceptual data model. The final step hereby referred to as materialisations of deductions will be the primary concern of this paper. In KnowID, favourability of materialising deductions in the knowledge layer is so that reasoning does not happen at runtime and this is suitable since conceptual models or ontologies do not change regularly [2].

Conceptual models can take on the form of Entity Relationship models (ER) used to represent the semantics of an application by employing entity constraints, attributes and relationships to represent a subject domain. We would then want to make this representation as expressive as can be so that many features of the reality being modelled can be interpreted [3].

There have been tools that have been able to implement materialisation of deductions of some formalised conceptual data models and ontologies such as an early version of Protégé and Intelligent Conceptual Modelling tool and methodology (ICOM). Whilst other tools like Crowd paired with reasoners have assisted users with the process with limitations.

Section 2 will discuss processes and the formalisation of ontologies necessary before reasoning. Section 3 will discuss all matters concerning reasoning including reasoning tasks and reasoners. Section 4 will discuss some tool implementations of materialisation of deductions and Section 5 will discuss an algorithmic approach for it. Section 6 will them discuss and compare the approaches mentioned and finally conclusions will be drawn.

2 Preliminaries

2.1 Priors to reasoning over a conceptual data model

For interpreting knowledge of a subject domain, an ontology or conceptual data model could be used to encapsulate the concepts of the subject domain and how they relate to each other. We must consider formally representing knowledge based on logic[2] in order to reason over it later. Before checking conceptual models for inconsistencies and abnormalities, the model must be formalised into a logic and then later sent to a reasoner that can query its properties [1]. There are however property-based differences between what is regarded as ontologies and conceptual data models. Conceptual models like Unified Modelling Language (UML), Object Relational Mapping (ORM) and (Enhanced) Entity Relationship model (EER) need to be formalised to express an ontology and do not have the reasoning capabilities of ontology languages like Web Ontology languages (OWL) which are rooted in logic [4]. However, it has been indication that both EER and ORM 2 can be formalised in Description Logics (DL). Description logics is a family of logics that are suited for representing information rooted in class structures [3]. EER can be formalised specifically in $\mathcal{ALNI}$ DL which can ensure tractable reasoning over it [4]. This is quite useful as EER has become most common for conceptually modelling relational databases [5].

For the purpose of this paper, the modelling language features of EER are the ones we look to explore in terms of being formalised in a logic. The features are: weak and strong entity type, n-ary relationship, attributes, basic cardinality constraints (0..n, 0..1, 1, 1..n) and identifiers, entity type subsumption, disjointness and covering constraints. DLR_{ifd} is the most appropriate logic for formalising these features [2]. DLR_{ifd} additionally encompasses identification constraints and functional dependencies [6] but unfortunately, there are no existing tools for this logic so we would want something close to this sort of logic. We also know that EER can be formalised in several different logics but they all employ OWA for automated reasoning [2].

The World Wide Web Consortium (W3C) designed OWL to formalise ontologies [7]. OWL is now a family of ontology languages that includes other iterations such as OWL 2 DL. An option we could take for formalising knowledge is having an ontology expressed in OWL 2 DL and then later using a DL reasoner for reasoning since there are tools for this. However features like n-aries and multi-attribute identifiers aren't catered to [2] but OWL 2 DL can otherwise be reasoned over [8] OWL files are difficult to manage and using them to process queries are not efficient. OWL is becoming increasingly useful to transform into EER models so it can be merged with relational databases easily. This is desirable because EER represents relational databases conceptually [5]. Rules have been developed to translate them into the EER model so that they map better onto relational databases and furthermore, there is also a methodology for mapping OWL full constructs to EER [5]. Bagui have shown how to map entities, attribute keys, relationships, super-classes and subclasses, cardinalities, as well as disjoint and covering relationships from OWL to EER. The methodology proposed here is intended to be implemented in case tools to automate this process [5].

3 Reasoning

When we consider the expressiveness of a language, the interactions between the different parts of the model become complex and issues of consistency arise [3] The more expressive and large it is.

3.1 Automated reasoning

We can have a language as a formalised subject domain but another engaging aspect is being able to reason over the language so that implicit knowledge can be deduced. Automated reasoning is about having systems automate the process of making inferences. This is done by implementing a formal language in which we can write conclusions and make assumptions about a problem as well as solving it, having implemented an algorithm to do so [9]. The idea is to have a number of premises that conclude to a hypothesis or generalize facts to an assumption. Computer scientists have done this formally in a logic, using rules and axiom as premises so that conclusions can be made in a depicted knowledge. It becomes tricky when we try to formally prove the conclusion we arrive at using the premises when we have a larger number of axioms [9] and so we look to automated reasoning. The explicit and implicit constraints of the conceptual model need to have automated reasoning done over them particularly when models become big and may have complicated interactions inside them [3] Primary considerations of automated reasoning include reasoning tasks and the formal language being reasoned over as well as choosing to optimise the algorithms that do the reasoning or restricting the language for lower complexity [9]. Reasoning tasks can include checking for consistency and classification of concepts in a hierarchy otherwise referred to as subsumption. The language being reasoned over may have fewer or more features like cardinality constraints to express the subject domain knowledge [9].

3.2 Reasoning tasks

3.1.1 Entity and relationship Subsumption
One of the reasoning tasks is checking to see if an entity subsumes another or likewise if a relationship subsumes another relationship for every legal database state [3]. If a class turns out to be a subclass of another and it is not explicitly implicated in the model, automated reasoning becomes essential [3].

3.1.2 Entity and relationship Consistency
An entity can be inconsistent in the sense that there is no database state where the entity has at least one object. Similarly, a relationship is found inconsistent if there is no database state where it has at least on object [3].

3.1.3 Satisfiability
Entity Satisfiability and entity subsumption are generally considered equivalent [3].

Many implementations of reasoning are restricted by computational complexity [9] Generally, we would have to choose between limiting the expressiveness of a language or otherwise trade off computational complexity. Description Logics is a logic that focuses on languages with good computational behaviour [9]. Ontology languages such as OWL and OWL 2 are based on description logics but lack expressiveness [9].

3.3 Reasoners

There are a number of automated reasoners that take in different languages and provide different reasoning services and thus may have different purposes. First-order and higher-order languages include Prover9, Vampire among others. DL reasoners used for OWL ontologies have these reasoning capabilities: satisfiability, consistency and instance classification (subsumption) [9]. One example of this is RacerPro.

3.3.1 Vampire is a tool that is a theorem prover for first-order (FO) logic but is flexible enough to be used as a reasoner for ontologies. First-order theorem provers and DL reasoners both provide reasoning capabilities but FO provers work with logic that is undecidable. FO provers use an invariable inference mechanism, meaning they are more flexible and can be tuned to different applications if processing algorithms are taken to task [10]. Vampire can check for consistency on ontologies but does not work well for large ontologies [10]. Vampire can deal with three datatypes namely, integers; real numbers and strings but it is not suitable for reasoning tasks such as satisfiability and in turn proving non-subsumption which is problematic because for most ontologies, concept pairs tend to be satisfiable and not in a subsumption relationship [10].

4 Tool implementations for Materialisation of Deductions

Materialization of deductions refers to editing a conceptual data model such as EER by adding all accepted inferences computed by a reasoner. And to verify the modifications made, a re-classification of it can be done where no new deductions are returned [2].

4.1 Crowd

Crowd is a web tool used for ontology engineering [4]. Crowd has multi-views and allows users to edit ontologies on a graphical platform with languages UML, EER and ORM. It provides DL reasoning for ontology engineering support [11]. The intention of this tool is to provide graphical methods to compose conceptual models and ontologies utilising logic-based reasoning services to describe a domain conceptually [1]. The main takeaways from this tool is that it allows modelling with graphical support, formalises models into logic-based formalisms and interacts with an off-the-shelf reasoner. It checks for satisfiability and implicit constraints which are then displayed using whatever language was being used as the initial graphical language. The first prototype has an OWLlink communication standard protocol that communicates with UML diagrams and the prototype also checks the satisfiability for the UML which is formalised to $\mathcal{ALCQI}$ description logics [1]. This also means users will get to see a graphical interpretation of the conceptual data model that was presented originally with the new deductions [1].

Crowd's server has an OWLlink translator that takes the graphical model as an OWL 2 file and sends it to the reasoner. It takes a JSON file of the conceptual model diagram and produces the equivalent OWLlink file. It has a module called query generator which sends queries to the reasoner. The reasoning is done by an off-the-shelf inference engine and results are given back to the front-end part of the system and shown [1]. When an inconsistency is found, it highlights the inconsistency and displays red traffic light.

The prototype currently uses the Racer DL reasoning server and allows one to create and edit UML diagrams. The reasoner checks for satisfiability for each class. The reasoner outputs false and highlights inconsistencies when a class/classes are unsatisfiable [1]. It is then up to the user to solve for the inconsistencies.

4.2 Early Protégé

Materialising deductions was available in an earlier version of Protégé [2]. Protégé offers a wide variation of modelling structures but inferencing is limited due to the OWL reasoners used for it. They are restricted to is-a relationships by using pulg-ins such as OWLViz and OntoGraf [1].

4.2.1 Protégé owl plugin

An OWL Plugin was extened for Protégé in order to modify OWL ontologies and make use of description logic reasoners. It provided services such as classification and checking for consistency [12]. The owl plugin can directly access DL reasoners like Racer. The user interface of the plugin supports reasoning services such as checking for consistency and subsumption. Users have the option to check consistency amongst all classes or selected classes. The protégé OWL API can access an API called Jena to infer any possibilities for subsumption. For files that are OWL full, they can be converted to OWL DL, otherwise only after an OWL DL file is checked for consistency will it then be checked for subsumption. The plug in will graphically show the subsumptions inferred in a hierarchy. Users will need to assert assumptions in order for them to be stored in OWL files but the two versions are shown side by side in the tool. [12].

4.3 ICOM

Intelligent Conceptual Modelling tool and methodology (ICOM) is a conceptual modelling tool that has a reasoner that is capable of confirming specifications, deducing inferences, re-evaluating constraints that can be stricter and detecting inconsistencies. It has a description logic reasoning server. The class diagrams are formalised using class-based logic. ICOM's conceptual modelling language is able to express an EER, entailing modelling features such as disjointness and covering constraints, classes (entities) and relations that can be defined as view expressions over other classes and relationships [13]. The tool goes beyond the implementation reasoning tasks subsumption and consistency. It also aims to employ more complex automated reasoning tasks to infer implicit deductions [13]. It Can deduce an inconsistent entity and can deduce an inconsistent relationship as well. ICOM is able to do all of this through reasoning with disjoint ness and covering constraints (partitioning of entities part of covering and disjoint Is-A hierarchy), cardinality constraints (mandatory participation, exactly one participation, at most one participation), subsumption of entities, subsumption of relationships. Through this same reasoning, it is able to edit cardinalities to make them stricter [13]. Additionally, it is able to deduce equivalence of classes [13].

The tool displays question marks where it finds inconsistencies, adds or edits arrows appropriately scripted by constraint numbers and labels for cardinalities, subsumption and equivalence. It's grounded in providing reasoning support for a graphical class-diagram that's deduction-complete. It gives users a deduction-complete ontology in graphical form after being given an ontology and reasoning over it to satisfy consistency, subsumption, cardinality and any other implied fact [13].

ICOM also makes a point to be interoperable by allowing one to import and export any ontology languages based on Description Logics and employs a mechanism to communicate with a description logic reasoner like RACER via its DIG protocol [13]. When using ICOM, the user must provide a URL of the DIG-suitable reasoning server and it does not have to be installed in the ontology editor. After the ontology is encoded in description logics, it is sent to the reasoner where class and association satisfiability are checked for, equivalance and subsumption amongst classes and associations, and calculating stricter maximum and minimum cardinality constraints, the system queries the system for deductions and results are given to the ontology editor to edit with the deductions. The editor displays unsatisfiable objects in red, non-explicit deductions are added in green, equivalance found is added using links, and likewise subsumption is shown using specific links and as well as stricter cardinalities added. The user can then decide if they want to keep the changes if they are happy with the deductions [13]. If ICOM detects a data reconciliation problem, where it may find a class that cannot have any instances, it displays a question mark on said class to notify user [13].

5 Algorithmic Materialisation of Deductions

Object programming languages are easier to use to modify an ontology than APIs like OWLAPI [7]. OWLready is a python module designed for ontology-oriented programming to support OWL 2 ontologies where entities can be accessed like objects in programming languages. A syntax is provided to manage classes and their constraints using this syntax, one is able to manipulate classes and do closed world reasoning. The syntax is based on Protégé's editor notations [7]. An algorithm to perform simple closed world reasoning tasks has also been implemented. OWLready can express

OWL 2 constructs such as classes, individuals (instances), disjointness, attributes (object properties) and datatypes. Methods are provided to map OWL to python object model in OWLready [7]. Python only uses an all-Disjoint mechanism as opposed to OWL that encompasses pairwise disjointness. OWLready also considers equivalence in the attributes of classes, checks for subclasses considering equivalent classes, features attributes in classes and supports datatypes boolean, float and string among more. OWLready provides functions to access and edit entities and constraints [7]. The user needs to call on the reasoner for inferences to be deduced using a sync_reasoner() function. It runs a HermiT reasoner on the file. The function returns the output produced by the reasoner and updates the ontology accordingly, modifying it in accordance to the inferences deduced by HermiT. Moreover, there is a closed_world function that will consider all classes and subclasses, and check for constraints to update which depend on existing relations and restrictions [7].

There are other management mechanisms employed for OWL like OWLLink. OWLLink is an interface that allows one to access OWL reasoner functionality [15]. OWLlink can be used to communicate with off-the-shelf reasoners [11].

6 Discussion

Bringing together visual modelling and reasoning is somewhat provided by ICOM and thus makes it a desireable end-user tool to use for materializing deductions. Reasoning services are provided by it including subsumptions m..n and n=1 cardinalities, disjunctions as well as equivalence whereas other tools do not currently have said architecture [4]. This suits some but not all of the features included in EER which would merge well onto relational databases. The approach ICOM takes in bringing together reasoning and graphical interpretation is not taken by Protégé-OWL editor. It instead makes edits on what is represented as a hierarchy of concepts in a panel and a hierarchy of properties in another panel and the deductions are shown in another tab panel not shown graphically. Moreover, the deductions panel is only provided for single concepts and not properties [13]. Moreover, ICOM employs more advanced and complex reasoning services compared to Protégé. ICOM intelligently may make considerations about lower and upper bounds about cardinalities through more complex reasoning than just is-A notions and inconsistencies that Protégé does. ICOM is so robust, it puts an isolated class in context and takes all constraints related to it into account. It has even been proved to derive all correct constraints [13] and his hence a suitable consideration for materialising deductions of a conceptual data model formalised in description logics. Furthermore, its deductions afford the ability of user validation. Once the derivation happens and the deductions are materialised on the ontology, the user can pick up their conceptual error or oversight and change accordingly [13]. ICOM uses DIG protocol instead of OWL API because the language is independent whereas API's are tied to a language like the OWL API being Java-based [13]. DIG protocol is however not actively developing and new reasoners are not supporting it.

Crowd's integration of graphical assistance and reasoning is not strong [1]. Crowd currently only provides means to edit UML with modelling features that include classes, attributes, generalisation and binary associations having n..m cardinalities when n, m>=1 [4]. The inference engine provides support for satisfiability checking when it comes to reasoning and the user can materialize and modify accordingly.

A worthwhile issue worth considering when it comes to editing ontologies is validation which is about the accuracy of ontology in its representation of the subject domain. Is it accurately interpreting the reality of the domain. It allows for the user to verify changes and cancel them if they don't represent the truthfulness of the domain. A problem with this is reversibility which is undoing the changes that have been made which is not always the same as inverting the change. An example is deleting a class in a hierarchy where the subclasses of that class must either also be deleted or reattached to a parent of the class. Reversing a change like this would require readjusting the hierarchy as opposed to just recreating the deleted element. Hence evolution logs can be used to manage such information about edits made [15]. Moreover, sometimes editing an ontology can introduce new inconsistencies. To prevent such undesired changes from occurring, a list of possible issues that may arise from changes should be shown to the user for their approval and be given the choice to cancel the changes and leave the ontology as is should they wish [15]. A tool like ICOM allows flexibility in this regard.

In the bigger picture, the problem with taking a user-interactive approach with materialising deductions and validating them goes against automating the whole process from formalisation to materialization of deductions so that the resultant conceptual data model is not generated at run time when used in conjunction or as a sub-process of a larger architecture. Hence, tools like Crowd would not work well for such an environment. Managing mechanisms such as OWLready would be better suited for such a task since it employs constructs in a module to manage OWL files and communicate with a reasoner to check for reasoning services and edit accordingly. We would ideally want the OWL to have features suited to EERs so that reasoning over it happens in closed world assumption.

7 Conclusions

This review looked at ways of interpreting knowledge expressed as an ontology or conceptual data model in a way that expresses the knowledge of a subject domain as much as possible. Doing this meant that implications often not explicitly shown in the model would need to be inferred and appended to the model. There have been some tools that have been able to do this given a model formalised in a suitable logic. Others have been able to make the inferences and leave it up to the user to edit the model accordingly if inconsistencies or subclasses were found. Additional to tools, a programming module was also created to automate this process by creating constructs to manage the file of the model and contact a reasoner to deduce inferences and edit accordingly with methods implemented in the module. Something important to note is that not any one implementation suited the features of those in the conceptual data model EER which are great for mapping of relational databases and posing queries that operate under closed world assumption.

REFERENCES

[1] Germán Braun, Laura Cecchi, Pablo Fillottrani, and Christian Gimenez. 2016. crowd: A Tool for Conceptual Modelling assisted by Automated Reasoning. In *Preliminary Report*, 2016, Argentina, 29-41.

[2] Pablo R. Fillotrani and C. Maria Keet. 2020. KnowID: An architecture for efficient Knowledge-driven Information and Data access. Universidad Nacional del Sur, Bah´ıa Blanca, Argentina. University of Cape Town, South Africa.

[3] Alessandro Artale, Diego Calvanese, Roman Kontchakov, Vladislav Ryzhikov, and Michail Zakharyaschev. 2007. Reasoning over Extended ER Models. In *26th International Conference on Conceptual Modeling (ER 2007)*, November 5 - 9, 2007, Auckland, New Zealand. Springer, Berlin, Heidelberg, 277-292.

[4] Germán Braun, Elsa Estevez, and Pablo Fillottrani. 2019. A Reference Architecture for Ontology Engineering Web Environments. *Journal of Computer Science & Technology* 19, 1 (April 2016) 22-31.

[5] Sikha Bagui. 2009. Mapping OWL to the Entity Relationship and Extended Entity Relationship Models. In *International Journal of Knowledge and Web Intelligence* 1, 1-2 (September 2009), 125-149.

[6] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. 2001. Identification Constraints and Functional Dependencies in Description Logics. In *Proceedings of the 17th international joint conference on Artificial Intelligence (IJCAI'01),* August, 2001, Seattle, Washington. Morgan Kaufmann Publishers Inc., San Francisco, CA, 155-160.

[7] Jean-Baptiste Lamy. 2017. Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies. *Artificial Intelligence in Medicine* 80, 11-28.

[8] Boris Motik, Peter F. Patel-Schneider. 2009. OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (September 2001). Retrieved May 1, 2020 from https://www.w3.org/2007/OWL/draft/ED-owl2-syntax-20090914/.

[9] C. Maria Keet. 2020. *An Introduction to Ontology Engineering*. Vol 20. College Publications.

[10] Ian Horrocks and Andrei Voronkov. 2006. Reasoning Support for Expressive Ontology Languages Using a Theorem Prover. In *Foundations of Information and Knowledge Systems 4th International Symposium (FoIKS 2006),* February 14 – 17, 2006, Budapest, Hungary. Springer, Berlin, Heidelberg, 201-218.

[11] Germán Braun, Laura Cecchi, Pablo Fillottrani, and Angela Oyarzun. 2018. A GraphicalWeb Tool with DL-based Reasoning Support over Orthogonal Variability Models. In *XXIV Congreso Argentino de Ciencias de la Computación (CACIC 2018)*, October 8 - 12, 2018, Tandil Argentina, 867-876.

[12] Holger Knublauch, Ray W. Fergerson, Natalya F. Noy, and Mark A. Musen. 2004. The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. In *Third International Semantic Web Conference (ISWC 2004),* November 7 – 11, 2004, Hiroshima, Japan. Springer, Berlin, Heidelberg, 229-243.

[13] Pablo R. Fillottrani, Enrico Franconi, and Sergio Tessaris. 2012. The ICOM 3.0 Intelligent Conceptual

Modelling tool and methodology. *Semantic Web* 3, 3 (2012), 293-306.

[14] Sean Bechhofer, Diego Calvanese, Matthew Horridge, Thorsten Liebig, Marko Luther, Olaf Noppens, Ralf Möller, Mariano Rodriguez , Michael Wessel, Evren Sirin, and Dmitry Tsarkov. 2008. OWLlink: DIG for OWL 2. In Proceedings of the Fifth OWLED Workshop on OWL: Experiences and Directions, collocated with the 7th International Semantic Web Conference (ISWC-2008), October 26 – 27, 2008, Karlsruhe, Germany.

[15] Alexander Maedche, Boris Motik, Ljiljana Stojanovic, Rudi Studer, and Raphael Volz. 2003 Ontologies for Enterprise Knowledge Management. *IEEE Intelligent Systems* 18, 2 (March-April 2003), 26-33.