Owl Monkey: Auto-Adaptive Application Builder for the Semantic Web of Things

Louis Bhérer,[a] Luc Vouligny,[b] Arnaud Zinflou,[b] Christian Desrosiers[a]
[a]*Information Technology Department, École de Technologie Supérieure, 1100 Rue Notre-Dame Ouest, Montréal, QC, Canada, H3C 1K3*
[b] *Institut de recherche d'Hydro-Québec, 1800 Boulevard Lionel-Boulet, Varennes, QC, Canada, J3X 1S1*

Abstract. Owl Monkey (OM) is an OWL auto-adaptive web application builder developed at Hydro-Québec. OM users are able to explore and build views on an OWL ontology and share those views without any prior knowledge of the technologies involved. Users can then do analytics over those views, use them to modify database content or export them. OM is highly generic and suitable for use with most OWL ontologies. Applications built with OM auto-adapt to modifications of the ontology and require no subsequent reprogramming. The overall effect is lower costs for application development and maintenance. Because of the genericity and auto-adaptability of this tool, this approach is a prime candidate for development of Industry 4.0 applications drawing on the Semantic Web of Things.

Keywords: Semantic Web of Things, OWL, Industry 4.0, Auto-Adaptive Application, Application Builder, Industrial Application

1. Introduction

At the dawn of the fourth industrial revolution, new ways of doing things call for new approaches to the design of computer applications supporting information systems.

In the Industry 4.0 paradigm, as data from all links in the value chain are becoming connected and transparently available, Semantic Web (SW) technologies[1] are increasingly explored as a means of harnessing the knowledge they contain [1]. Their reasoning, interoperable and auto-adaptive capabilities make them a prime choice for addressing problems involving cyber-physical systems.

In efforts to reach its full potential, Industry 4.0 is seeking to, amongst others, facilitate rapid deployment of applications, develop a holistic engineering approach for re-engineering with minimum effort and simplify use of those applications/approaches by users new to the underlying technologies [2].

From this perspective, this paper describes an industrial application of the semantic technologies at Hydro-Québec (HQ),[2] the provincial power utility that generates, transmits and distributes electricity. This industrial application stems from research under way on knowledge management and evolutive prototyping [3, 4, 5, 6] at IREQ, the utility's research institute.[3]

Owl Monkey[4] (OM) builds views that can be used and shared by lay users of semantic technologies and query languages in general. OM allows browsing ontologies in Web Ontology Language (OWL) to discover the knowledge they contain by iteratively combining the data of its different classes. To do this, OM uses generic queries that simultaneously interrogate model and data.

[1] https://www.w3.org/standards/semanticweb/

[2] http://www.hydroquebec.com/residentiel/
[3] http://www.hydroquebec.com/innovation/fr/institut-recherche.html
[4] https://vimeo.com/album/5826057

This approach makes applications auto-adaptable to changes in the data model. Modifications to the ontologies are immediately visible to all users without the need to reprogram or recompile the application. This auto-adaptability is especially useful in a context of evolutive prototyping, Agile method or constant changes in the value chain as foreseeable for Industry 4.0.

The main contributions of this paper are as follows:

1- It offers an approach for *ad hoc* combination of the data of an OWL ontology.
2- It offers a process for building auto-adaptive applications that minimize the effort of rollout and maintenance.
3- It offers a knowledge discovery approach that is highly generic to OWL ontologies.
4- It offers intuitive semantic data visualization and manipulation tools that can be used by semantic technology neophytes.

2. Literature Review

The world we know is at the beginning of the fourth industrial revolution. Industry 4.0, as this revolution is called, consists in integrating data resulting from an exponential increase in the number of sensors at all stages of the supply chain and from the interconnectivity of cybernetic devices. Industry 4.0 may be defined as a set of technologies that are transforming how systems connecting physical assets to calculating capacity are managed [7].

It rests on nine pillars representing technologies for connecting and analyzing data, predicting future states of the system and reconfiguring value chain components [7]. Two of those technologies are relevant to this research, namely, the Industrial Internet of Things (IIoT) and horizontal and vertical systems integration.

The systems integration pillar implies that all organizational systems are interconnected and that organizations themselves are interconnected as well.

The Internet of Things (IoT) technology refers to a system of interconnected objects. IIoT is the IoT application in an industrial context. Thanks to this interconnection, the value chain components can interact among themselves and with the environment, thereby easing rapid adaptation to change [8].

The Semantic Web of Things (SWoT) is explored as a promising avenue for facilitating the interoperability of these systems by allowing for the integration of heterogeneous objects through semantically rich information [1]. SWoT is the integration of SW technologies with the IoT [9].

SW is a W3C [5] initiative intended to build a technology stack for moving from a Web of Document web to a web of Data. This technology stack includes Resources Description Framework [6] (RDF), SPARQL Query Language for RDF [7] (SPARQL), Resource Description Framework Schema [8] (RDFS), and OWL, [9] among other technologies.

RDF is a standard model for data exchange on the Web and takes the form of a directed, labeled graph. It is based on Uniform Resource Identifier (URI) technology, a character chain for unique, permanent resource identification.

SPARQL is a query language for interrogating data in RDF format.

RDFS is a knowledge representation language for structuring the RDF resources in ontologies.

OWL is a language used over RDFS to represent more complex knowledge about objects, groups of objects or relationships between objects. RDFS and OWL are based on first-order logic and allow the use of inference engines to explain the implicit information in the data and to confirm their logical coherence.

RDF data can be stored in triple databases (TDB). A triple is the basic unit of the RDF language and consists of three parts: subject, predicate and object. Subject and predicate are URIs representing, respectively, the resource concerned and its relationship with the object. The object is a URI or a literal and may therefore represent a resource or a value, respectively.

Many Industry 4.0 requirements can be addressed by SW technologies. The authors of [10] propose the use of ontologies in the factories of tomorrow given their auto-organization, auto-learning and auto-adaptation properties.

Whereas IoT is meant as a system of interconnected objects, the Web of Things (WoT) is intended to integrate those objects at the web level. SWoT is meant to draw on SW and WoT to facilitate the creation of a knowledge system by

[5] https://www.w3.org
[6] https://www.w3.org/RDF/
[7] https://www.w3.org/TR/rdf-sparql-query/
[8] https://www.w3.org/TR/rdf-schema/
[9] https://www.w3.org/OWL/

adding meaning to the data and simplifying the interoperability of cybernetic systems [11].

It seems desirable for Industry 4.0 information system applications drawing on SWoT to auto-adapt to the data models of objects. The two key concepts for achieving this flexibility are the genericity of the applications and their auto-adaptability to the data model.

Genericity is defined here as the capacity of an application to function with knowledge from any domain. Where OM is concerned, this means its capacity to function with almost any OWL ontology. Auto-adaptability is defined as the capacity of the application to auto-adjust to the model content upon changes in the data model without having to modify or recompile its code.

In [12], McGinnes and Kapros circumscribe the problem of the non-adaptability of applications as a conceptual dependence to the data model. They describe this dependence as undesirable software coupling. Those authors conclude that most information system-based applications are dependent on their data model. Consequently, those systems need maintenance whenever the data model is modified. McGinnes and Kapros propose six principles to achieve conceptual independence for any data source. Bhérer et al. explain in [13] how RDF-based technologies intrinsically comprise several of these principles.

There is a two-fold advantage to achieving conceptual independence. First, it provides for auto-adaptive applications. Second, the client-server tiers of such applications are actually a collection of generic components that can be used with most ontologies, regardless of the domain they describe [13]. This makes it possible to generate new applications by assembling those components around domain ontologies.

There are a few commercial solutions that attempt to harness the evolutivity and flexibility of semantic technologies in application building. TopQuadrant[10] offers a complete array of products using semantic technologies. To take a few examples, TopBraid Suite[11] implements SPARQL Inferencing Notation (SPIN), [12] a language for representing SPARQL rules and constraints that has meta-modeling capabilities for integrating SPARQL functions, as well as query canvases in semantic models. The SPIN technology stack is extended by SPARQL Web Pages (SWP), [13] an RDF-based development framework for composing user interfaces to visualize semantic data. SWP has the advantage of integrating (a) dynamic SPARQL queries and (b) presentation of the results as HTML, SVG, JSON or XML code in the data model.

inova8[14] is aimed at encouraging enterprises to use linked data by proposing solutions that facilitate exchanges between software components. OData2SPARQL includes a protocol for creating and consuming applicative programming interfaces based on REST methodology [14]. inova8 proposes using their technologies for the rapid construction of applications having some auto-adaptive capabilities.

As for graphic interfaces, some authors [15, 16] agree that SW requires new visualization and interaction mechanisms to manipulate its data. Besides the challenges of visualization in general [17], SW data have challenges all their own [18].

The authors of [18] see one of those challenges as the need for a semantic visualization solution wherein users could work with linked data without having any technical knowledge. They should be able to understand data structure, implicitly build queries, identify links between resources and discover new knowledge.

Theoretically, as was mentioned in [19], expressivity and usability are the main two criteria when various alternatives fare compared with query specification techniques for semantic research. Expressivity represents the variety of queries that a system can pose, whereas usability is the ease with which a user learns interface efficiency and intuitivity.

OM falls in the category of Visual Query Systems (VQS). A VQS is a tool designed to simplify the query language in order to protect users from its underlying complexity. This lessens language expressivity but increases usability of the tool and thus the time it will take a neophyte to master the tool [20].

A VQS must support the two complementary activities of exploration and building. Because of the nature of semantic data, the most commonly used visual representation to do so, is graphic representation, although this approach is criticized [17, 21]. Form representation and facet representation are also used extensively, facets being orthogonal conceptual dimensions partitioning the research space [22].

[10] https://www.topquadrant.com/
[11] https://www.topquadrant.com/products/topbraid-enterprise-data-governance/
[12] http://spinrdf.org/
[13] http://uispin.org/
[14] http://inova8.com/bg_inova8.com/

3. Hydro-Québec's Information Systems

HQ is the main electricity generator in Canada, having 63 hydroelectric power plants and installed capacity upwards of 37,000 megawatts. It had some 4.3 million customers in 2018. HQ has a huge number of corporate systems. To give some idea of the magnitude, the grid control system alone is based on more than 80 corporate systems.

One example of an HQ system is the Système de Commande et de Surveillance en Centrale (SCSC), which is the power plant control and supervisory system giving access to Hydro-Generator Units (HGU) monitoring data. Various tools use SCSC to analyze the behavior of HGU and ancillary systems, detect anomalies, make diagnoses, assist with commissioning and energizing HGU, help with systematic, conditional and predictive maintenance, and more.

Supervisory Control and Data Acquisition (SCADA) of hydrometric data is another example of a corporate system. SCADA is used to evaluate the water resources available for the power plants. It comprises several hundred water level gauges (limnimeters), gate opening sensors, voltage sensors, sensors for real and reactive power, flow sensors, etc. It, too, is the source of several corporate analysis and prediction tools.

OM was used to develop an application connecting with the system called Étude du Vieillissement de l'Appareillage (EVA) in French. This system for assessing equipment aging is designed to help electricians with routine servicing of electrical equipment. For one, it provides for logging their test results in a database. Several computer tools retrieve the necessary information from EVA as input for optimal maintenance of power transmission assets. EVA is also used to make statistical calculations for certain types of tests, to detect anomalies and follow up on them, and to visualize, standardize and export data.

EVA interacts with other corporate systems, including MAXIMO, an inventory bank for all HQ equipment. EVA comprises a considerable number of decentralized databases (DB) of three types: operational DBs for electrical test data, regional DBs for oil tests and aggregations of operational DBs, and a provincial DB for standards and aggregations of regional DBs.

An IREQ research group sharing in a maintenance project produced formulas for characterizing the extent of transformer insulation deterioration from chemical compounds in the insulating oil. Protocols for measuring the presence of those compounds were developed and implemented in the field. At this point, transformer oil test data is accessible through EVA. These are the data in the use case below.

4. Use Case

OM was used to build several applications at Hydro-Québec, most of them as prototypes and one of those being rolled out at this time.

The main SWoT-based application was developed as part of a project for electric transformer diagnosis and analysis. It is called DATE in French (Diagnostique et analyse des transformateurs électriques). This application is used to register and share research data produced by IREQ's chemical researchers.

Based on measurements of chemical compounds in the oils, formulas determine how far power transformer insulators have deteriorated. This approach has proved to be a cost-effective alternative to costly inspections that require opening the transformer to sample the insulation. Those measurements are stored in EVA and must be retrieved to diagnose the transformers and schedule their maintenance.

Thanks to D2RQ[15], a library for mapping a relational DB in RDF format, among other uses, a correspondence file was created to download EVA data in the OM TDB. The model contains half a dozen or so classes and some 40 properties.

The retrieved EVA data is enriched with the results of the calculations for determining molecule concentrations and classifying the transformers. The researchers also use OM to integrate measurements done on personal samples in the TDB.

Once the model has been created and the data imported, users then have at hand all OM functionalities for exploring data and for building and sharing views.

5. Owl Monkey

OM was developed as a three-tier application. The client interfaces are programmed in JavaScript[16] by means of Sencha's Ext JS development

[15] http://d2rq.org/
[16] https://fr.wikipedia.org/wiki/JavaScript

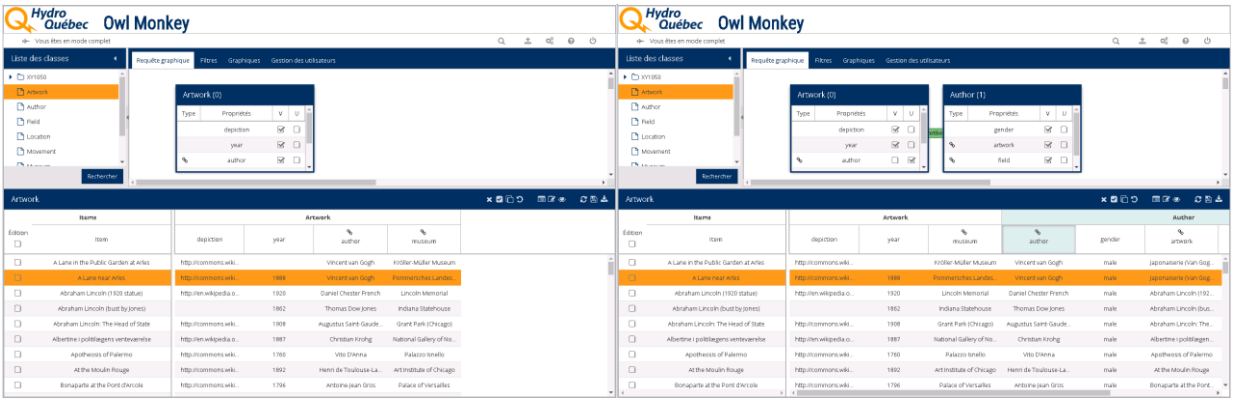

Figure 1: Subject class only (on left) and subject class with linked class (on right)

framework [17]. Client-server communication takes place through REpresentational State Transfer (REST) services [15]. OM uses the Spring [18] development framework on a TomCat 7 server [19] to manage services and user rights. The server side is developed in Java 8[20]. It communicates with the Oracle 12c database [21] equipped with the Spatial and Graph module [22], via SPARQL queries launched from the Jena library [23]. Spring, TomCat, Java and Oracle were selected because they are part of the standard software used at HQ.

OM is a web tool used for both exploration (finding new knowledge) and ontological exploitation (using existing knowledge) [23]. It is used to build views for exchanging information between stakeholders and modifying the underlying knowledge bases. OM was designed for use by SW technology neophytes and data manipulation languages (DML) in general. Some degree of SPARQL expressivity was sacrificed for improved usability. That usability is attributable chiefly to the use of widespread visualization paradigms and simple interaction mechanisms. The main tool of the application is a two-dimensional grid representing a graph of linked data.

5.1. OM usability

The semantic grid is the center of OM, in which the graph of class, its class properties and its individuals take the form of a two-dimensional table.

The first class loaded from a list of ontological classes becomes the subject class of the view, the term subject class denoting that the view applies to the individuals of the class in question. Consequently, the subject class is the starting point for the query, and all other classes that will be part of the view will be connected to the subject class by a pattern or set of patterns. The term pattern here refers to a triple in which variables have replaced subject and/or predicate and/or object.

The properties of an ontological class are represented by the columns of the semantic grid. It follows that those columns are also facets of the solution space. All of the properties shown are of two mutually exclusive types: data properties (owl:DatatypeProperty) and object properties (owl:ObjectProperty).

Data properties are RDF graph leaves containing literal values. OM uses datatypes (modeled in the property range by rdfs:range) of those literals to ensure appropriate presentation of the values and adapt the interface components, e.g. filters or input fields.

Object properties are graph branches relating individuals of one class with those of another class. A functionality has been implemented whereby users load the properties describing the range class of that property (defined by rdfs:range) in the grid, also loading the values of its individuals related to the individuals of the subject class. Figure 1 illustrates this functionality. The subject class before use of that functionality is shown on the left, whereas the subject class and the linked class are seen on the right.

This enables users to browse the branches of the graph and explore the ontology. In browsing the links between the classes, users implicitly compose a query and keep the query solution space in sight at all times.

[17] https://www.sencha.com/products/extjs/
[18] http://projects.spring.io/spring-framework/
[19] https://tomcat.apache.org/download-70.cgi
[20] https://www.java.com/fr/
[21] https://docs.oracle.com/database/121/RDFRM/toc.htm
[22] https://www.oracle.com/technetwork/database/options/spatialandgraph
[23] https://jena.apache.org/

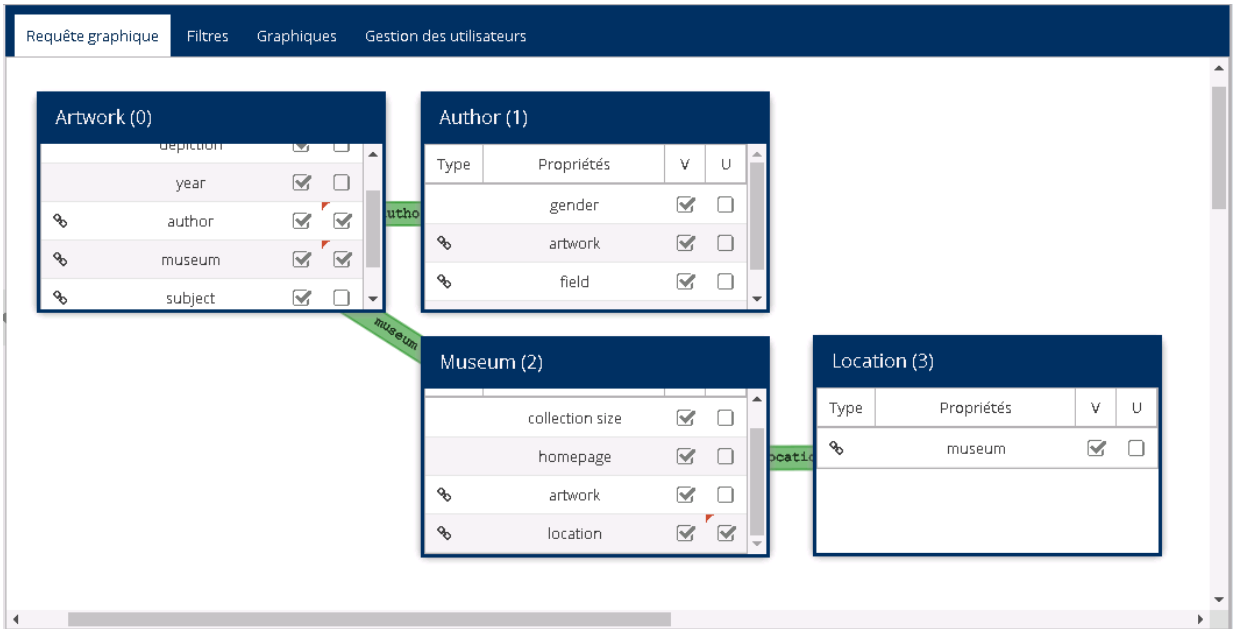

Figure 2: Graphic query

The semantic nature of the data explains how the view classes can be loaded independently one from the other. In fact, the grid is populated using URIs to align the individuals of non-subject classes with individuals of subject classes. Because of this lazy class loading, the same set of generic queries can be used for *ad hoc* addition of a class to the solution set.

Several functionalities of the semantic grid support view building. Columns and column groups headers can be dragged and dropped to modify the order of class properties and the order of classes. Additionally, each column contains a contextual menu to sort the grid according to the values in that column or to hide, show or filter the dataset with the facet of this column.

When an individual in one class is related to multiple individuals in another class, a contextual menu is used to navigate among the possibilities. That contextual menu is displayed by clicking on a cell in an object column. Choosing an individual from that menu displays the values for that individual and the related individuals in the corresponding line of the grid.

Since each individual in a class can be related to multiple individuals in another class, there are functionalities that clone or hide the lines. This ensures that users can display the full content of the graph in one single view if they wish.

Among the many tools that could be regarded as a VQS studied in this research project [21, 22, 24, 25], only [25] can be used for direct building of the solution space in a two-dimensional grid. However, [25] does not allow the iterative combining of classes and apparently cannot be used on more than one class at a time.

There are two advantages to building a view dynamically in one single grid through exploration and iterative combining of data from several classes. First, the grid is a widespread visualization paradigm with which most users are familiar. Second, it is used to explore the data model, while having a resulting view that is constantly in sight without the need to change perspective, as is the case with such tools as [21].

OM has a simplified mode for users who do not need to build views. The simplified mode gives access to fewer application functionalities and is intended for the use of existing views only. Still, users of the simplified mode are able to apply filters and modify TDB data. In sum, this mode extends OM to use cases requiring greater usability.

The OM approach equips end users to create the views they want through *ad hoc* exploration of domain data. This seems a relevant approach in the Industry 4.0 context, in which the search for flexibility and reactivity tends to put some of the decision-making power back in the hands of domain experts.

Furthermore, this kind of approach could shortens the time that computer experts spend developing and maintaining applications by shortening the time for determining the clients' functional needs and

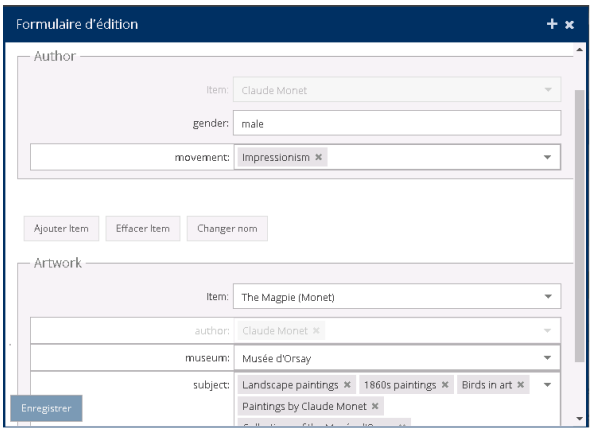

Figure 3: TDB editing tool

leaving it to domain experts to build and modify the application views.

This approach differs from an approach such as SWP, where ontologists are left to define complex data relationships and dictate how data is to be represented on the user interface. Although SWP gives ontologists more to work with and could make the representation clearer, it implies that users unfamiliar with semantic technologies cannot readily use such a language or modify applications as they wish. Nor is it clear that the SWP approach can be used for intuitive exploration of data models. As for inova8, it uses application canvases that can be edited by a programmer, and this does not allow end users to build views as they want.

A tool to visualize a graphic version of the query under construction is available in OM. But if users prefer, they can use those graphic components to explore the ontology and build the solution space. Figure 2 shows a query created from four classes. The use of graphic representation in OM was chosen to give users a mental picture of the query they have formulated. In some cases, different queries lead to very similar grid representations. Graphic visualization in these cases helps recognize the query structure at a glance.

OM also uses visualization tools in form format (Figure 3). Users can use them to create, modify or delete information on individuals involved in a line selected from the TDB grid.

These forms are built dynamically based on the visible columns of the grid and the column order. The nature of the required fields is determined by the ranges of the properties represented by these columns.

The cardinalities implied by the functional properties determine whether the object fields will be represented by comboboxes allowing only one possible value at a time or tagfields for inputting as many values as users want.

The label fields for all non-subject classes are programmed to relate the object field of the preceding class, which serves as a bridge between the two classes. Any change in that field is also applied to the label field and the choices available in its combobox. The same is true for inverse type properties that potentially relate to fields of the preceding classes.

Form representation is another known intuitive interaction mechanism for users. It gives another perspective on the graph data and shows in real time the consequences of user-requested modifications.

5.2. OM genericity

After usability, genericity dictated the design guidelines for the application. OM genericity is obtained by building on the inherent characteristics of the RDF, RDFS and OWL standards [26]. Given that all of its queries are generic, OM can work with most OWL ontologies that adhere to certain basic principles.

At this point, OM works only with ontologies with properties having one domain and one range. This puts limits on OM genericity, but several known ontologies, as in DBpedia,[24] GoodRelations[25] or Common Information Mode,l[26] fit this hypothesis. Working with ontologies that do not meet this criterion could be possible through semi-automatic mechanisms (e.g., proposing that users choose the class they wish to add to their view when an object property has several ranges) or through automatic mechanisms (e.g., simultaneously adding all possible classes to the view). There will be further research on these possibilities.

Generic queries are built by simultaneously interrogating model and data. A query is called generic when its pattern contains only variables or elements of the modeling languages, i.e., in OM: RDFS, OWL and the visualization ontology (VO) discussed further on. In OM, the visualization model and information transit in graph form between TDB and client. This makes it possible to interrogate the ontologies regardless of their domain.

OWL and RDFS expressivity occurs in their semantics, which contain significant information retrieved by generic queries and used by OM to

[24] https://wiki.dbpedia.org/
[25] http://www.heppnetz.de/projects/goodrelations/
[26] https://cimug.ucaiug.org/

adapt its interface and guide users within the logical limits of the model.

To take an example, the TDB data modification form limits user actions by the nature of the fields, and this depends on property cardinality, range or type. The same is true of filters and graphs (barchart, piechart, etc.) that OM offers users.

Annotation properties, too, are used to format and valorize the different application components. For instance, URIs are systematically replaced by labels (rdfs:label), and comments (rdfs:comment) are found in tooltips.

OM genericity has a number of advantages. To begin with, it accelerates application building since the logical components adapt to the knowledge in the OWL ontology. Model production and data import may therefore be the only stages requiring information system development upon building an application.

Applications created with OM also have a kind of auto-adaptivity. As generic queries interrogate the model upon each query, changes in the model are immediately reflected in the applications with no need to reprogram or recompile. This ensures that applications continue providing adequate information when the ontology evolves.

These properties seem to be highly useful in the Industry 4.0 context, where the aim is to combine equipment data that is part of a steadily evolving value chain.

5.3. Visualization ontology

VO drives the display of components based on the data model annotations it allows. It also contains semantics describing the views saved with OM. This ontology contains no class or individual, only properties. These are four property types: class annotation properties, property annotation properties, views annotation properties and meta-properties.

VO annotation properties may indicate that a class or a property can be used in certain OM functionalities. An example would be indicating that the label of individuals in a class may or must be auto-generated based on the value of certain properties.

They can also be a framework for user actions. They could, for example, indicate the minimum or maximum of numeric properties or stipulate that the individuals in a class must have a value combination unique to certain properties.

Lastly, they could give users more information about the model, by indicating the unit of a property, for instance.

In an approach such as OM, all information that ensures the dynamicity of the interface components must be included in the data model in order to keep the applications generic. VO has made certain OM functionalities generic to VO-annotated ontologies.

In the Industry 4.0 context, development of a standard VO for annotating devices in the value chain would seem to have some advantages. For one, the models could contain information needed for auto-adaptive applications to provide complex functionalities adapted to those devices.

VO also contains annotation properties to persist all the information needed to load a view given that saved views in OM take the form of RDF graphs in the TDB.

The genericity of OM queries makes it possible to load views created on a data model that has evolved since it was saved. Several types of changes in the ontology are immediately reflected in the views without any modification from queries underlying the saved views. For example, changing the label or comment of a property or class, adding a property or class, changing the cardinalities of the properties will be reflected in the views without incident, provided they do not logically invalidate the TDB.

Modifications such as the removal of properties or classes may, however, modify the meaning of saved views. VO saved annotations could then be used to determine which views are impacted by modification of the ontology and how that modification affects their meaning. By way of response, semi-automatic and automatic mechanisms could, respectively, notify the view builder of the modification and propose possible solutions or update the saved views affected when possible.

VO also contains properties applicable to all individuals in the TDB. These are not annotation properties like the other VO properties. Rather, they are data properties and are associated by inference with all classes of the model.

Those properties ensure that modifications to the TDB are followed up. Every individual added to the TDB is automatically assigned a value for each of those properties to tell which user created or modified the individual and when. Those properties contribute to data quality and validity through an accountability mechanism. They also provide for attaching permission to the views so that only the

creator of an individual can modify the values involved.

5.4. Use of inference

OM was designed to take advantage of the inference engines and power of the RDFS and OWL languages. Because of RDFS and OWL expressivity, inference engines can deduce information implicitly contained in the data. Each of these languages comes with a set of entailments for validating the logical coherence of the assertions in the graph and deducing additional assertions.

In a context such as Industry 4.0, where *ad hoc* unions between data models would be desirable, inferences based on first-order logic could prove advantageous. For example, applications could use inferences for automatically deducing that a device belongs to a class of devices, identifying correspondences between individuals from different models or testing the logical validity of the union of ontologies.

In the development stage, inference engines allowing forward and backward chaining were tested. Backward chaining implies that inference is applied to each query solely on the resources concerned by that query. Forward chaining is applied on the entire TDB before queries are made.

In an application such as OM, where users can constantly modify the data and the model itself, backward chaining could be a suitable choice. Given, however, that the use of Oracle represents a corporate constraint and the Oracle inference engine does not natively allow for backward chaining, OM was developed for forward chaining.

There is a big issue with forward chaining in the OM context in that it requires redoing the inference throughout the dataset after each triple suppression. The wait time for data modification operations becomes prohibitive.

It was necessary to develop contrivances to avoid redoing the inference whenever data was modified. This has allowed the use of forward chaining for all use cases encountered to date.

5.5. Classification and heritage paradigms

Owing to how the OM interface presents ontological classes and the properties associated with those classes, users can understand the nature of the individuals in a class and add new individuals. The classification paradigm, in which we find the RDFS and OWL inference sets, and the heritage paradigm of object-oriented languages differ as to how properties are matched with the classes in the model.

In a heritage paradigm, the properties applicable to a class are the properties of that class and its superclasses.

In a classification paradigm, an individual's type is deduced from the properties of that individual and from the classes whose domain includes those properties. It can be deduced, for example, whether an individual is a living organism, a mammal or a human by finding the properties associated with the most specific subclass of the hierarchy. Given this, in seeking the properties that have a given class as the domain, the properties of that class and its subclasses will be found once RDFS and OWL inference sets have been used on the data.

It seems better for the interface to have an object-oriented vision of a class (i.e., the properties of that class and its superclasses) since the superclass properties may be needed to understand the nature of an individual. One could also consider presenting both an object-oriented vision and a classification vision of the properties of a class, i.e., the properties of that class, its superclasses and its subclasses.

Several factors enter into choosing between the object-oriented vision and the mixed vision. With the former, the user will not see all the properties of the individuals. Although this may not be desirable, it nevertheless makes it easier to understand the information in the grid. With the latter, all information in the model is presented. Although this may cause information overload, users have available information for understanding the individuals.

Also to be considered is the constraint on use of the tool implied by the choice of vision. In the first case, the interface constrains users to create individuals in the class selected, but prevents them from creating individuals in its subclasses without modifying the view. In the second case, users may create individuals belonging to a subclass of the class selected, but also individuals belonging simultaneously to several subclasses. This may not be a desirable option, even if mitigated through the use of semantics to declare disjoint classes.

A similar situation arises for property ranges. In a classification paradigm, a property is found after inference with several ranges – its original range and subclasses of that range. On the client side, the present version of OM requires differentiating between the original range and the inferred ranges

	If	Then
pi-rng	T(?p, rdfs:range, ?c)	T(?p, om:hasDirectRange, ?c)
pi-rng-inv-1	T(?p1, owl:inverseOf ?p2) T(?p2, rdfs:domain, ?c)	T(?p1, om:hasDirectRange, ?c)
pi-rng-inv-2	T(?p1, owl:inverseOf ?p2) T(?p1, rdfs:domain, ?c)	T(?p2, om:hasDirectRange, ?c)
pi-dom-1	T(?p, rdfs:domain ?c)	T(?p, om:hasDirectDomain, ?c)
pi-dom-2	T(?c1, rdfs:subClassOf+, ?c2) T(?p, rdfs:domain ?c2)	T(?p, om:hasDirectDomain, ?c1)
pi-dom-inv1	T(?c1, rdfs:subClassOf+, ?c2) T(?p1, rdfs:range, ?c2) T(?p1, owl:inverseOf, ?p2)	T(?p2, om:hasDirectDomain, ?c1)
pi-dom-inv2	T(?c1, rdfs:subClassOf+, ?c2) T(?p1, rdfs:range, ?c2) T(?p2, owl:inverseOf, ?p1)	T(?p2, om:hasDirectDomain, ?c1)

Table I. Pre-inference set

so as to determine which class must be retrieved when a user decides to explore the links of an object property. This original range is the most specialized class of all the ranges, assuming there is only one original range. As was mentioned earlier, future versions of OM could allow users to consider different property ranges and choose the class to add to the grid. This would allow them to select any subclass of the hierarchy.

Regardless of the vision chosen for the class properties, a way is needed to find the properties of the superclasses of the selected class, with queries on the model, not the individuals.

OM is used to explore linked data by using the ontology, not the individuals, to manage exploration, presentation and use. Several authors recommend this technique because it helps with interpreting content and identifying implicit and explicit links between the data [16]. In OM, this provides a means of presenting classes having no individuals, as well as never-used properties, so that users can add such information to the TDB.

Two solutions have been contemplated for obtaining object-oriented visions of class properties. The first consists in determining the properties whose domain consists of the observed class and all its superclasses by means of a SPARQL query upon each user request. At the same time, this determines which class among those that are ranges of those properties is the most specialized in the ontology. This approach has the advantage of allowing interrogation of datasets already inferred, e.g., those found on endpoints outside the enterprise. However, the complexity of the queries increases the reply wait time.

The alternative solution used in OM is to have a set of rules inferred before using RDFS and OWL inference sets. This pre-inference set of rules is shown in Table 1. The term directDomain indicates the domains of a property in a heritage perspective, whereas directRange identifies the most specialized of the ranges. This ensures that 1) each property has one single directRange, 2) directRange corresponds to the most specialized of the hierarchy classes that can contain that range, 3) each property has as directDomain the observed class and all its superclasses and 4) the inverse properties (that have yet to be processed because the RDFS and OWL inference sets are used only afterwards) are considered in attributing directDomains. Inference can then be addressed through RDFS and OWL rule sets. The directDomain and directRange semantics are then used in the generic queries.

This solution was implemented for ontologies defining only one domain and one range for each property prior to inference. It can, however, be adapted for application to ontologies that do not fit this assumption in accordance with the mechanisms developed to adapt the interface for this issue, as was explained earlier. The advantage of using inference to define the direct domains and ranges is that this will accelerate the generic queries.

6. Discussion

OM aims to occupy a unique niche in the corporate toolbox of Industry 4.0 by capitalizing on the advantages of semantic technologies.

Corporations the size of Hydro-Québec have tremendous need for information systems. OM is

primarily meant to help with rapid application development.

Applications are built with OM in three stages:

1. Creating data models.
2. Importing data in OWL format.
3. Building and sharing views.

In the applications built with OM to date, the data model has been created from the existing data and research needs to come. However, several popular OWL ontologies work with OM, including DBpedia, GoodRelations and Common Information Model, and could be used as the basis for building applications.

Once a preliminary version of the data model has been created, meetings are held with stakeholders to adjust the model nomenclature and structure. Because of the auto-adaptability of OM, such changes can be made at any point in the development process without requiring additional development.

Data importing is contingent upon the data source. It may consist, for example, in organizing the data in CSV format and importing it into the TDB using programs such as Cellfie,[27] a Protégé plug-in.[28] As for the DATE project, it consisted in programming a connector linking a relational DB and the OM TDB.

Lastly, the tool is put in the hands of the domain experts who make the views that will be shared.

Calling on domain experts to compose, modify and manage views gives them more leeway and the possibility to explore the data and reorganize the application as they see fit. This puts some of the responsibility for decisions back in the hand of the resources affected by the decision, which may boost corporate reactivity.

The chief advantage of applications created with OM is their auto-adaptability. In a corporate context, adaptability to change, whether through science, markets or administrative decision, is a critical factor in the global reactivity of enterprises.

The models used in the applications created by OM can evolve while those applications continue displaying adequate information. As a result, new knowledge elements can be used immediately and shared instantaneously without the need for a formal modification procedure.

This auto-adaptability makes OM an interesting option in the Industry 4.0 context, where *ad hoc* data combination among several models is desired, without those models being known beforehand and the applications created auto-adapting to constant system reengineering.

[27] https://github.com/protegeproject/cellfie-plugin/wiki
[28] https://protege.stanford.edu/products.php

7. Conclusion and Future Studies

In the corporate world, the rollout of applications created with OM shows that an auto-adaptive approach using semantic technologies is not only possible, but has the potential to sweep remarkable synergy in its wake.

Looking at future studies, several new functionalities are being considered in the short term. This includes data model editing tools for users and developing mechanisms for automatic notification upon changes in models impacting the views.

Other than those changes, studies contemplated over the longer term are concerned first with OM capacities for automatic match-up of individuals from different ontologies. The application seems to have a strong potential in allowing end users to cross data from different OWL models. Because each class is loaded independently of the other classes, there is a possibility of making generic queries on different TDBs in order to populate one single grid.

8. Bibliographie

1. Vogel-Heuser, Birgit, and Dieter Hess. "Guest editorial industry 4.0–prerequisites and visions." *IEEE Transactions on Automation Science and Engineering* 13.2 (2016): 411-413.

2. Thuluva, Aparna Saisree, Darko Anicic, and Sebastian Rudolph. "Semantic Web of Things for Industry 4.0." *RuleML+ RR (Supplement)*. 2017.

3. Zinflou, Arnaud, et al. "Application of an ontology-based and rule-based model in electric power utilities." *2013 IEEE Seventh International Conference on Semantic Computing*. IEEE, 2013.

4. Gaha, Mohamed, et al. "An ontology-based reasoning approach for electric power utilities." *International Conference on Web Reasoning and Rule Systems*. Springer, Berlin, Heidelberg, 2013.

5. Vouligny, Luc, and Jean-Marc Robert. "Online help system design based on the situated action theory." *Proceedings of the 2005 Latin American conference on Human-computer interaction*. ACM, 2005.

6. Vouligny, Luc, Claude Hudon, and Duc Ngoc Nguyen. "Design of MIDA, a Web-Based Diagnostic Application for Hydroelectric Generators." *2009 Fourth International Multi-Conference on Computing in the Global Information Technology*. IEEE, 2009.

7. Rüßmann, Michael, et al. "Industry 4.0: The future of productivity and growth in manufacturing industries." *Boston Consulting Group* 9.1 (2015): 54-89.

8. Vaidya, Saurabh, Prashant Ambad, and Santosh Bhosle. "Industry 4.0–a glimpse." *Procedia Manufacturing* 20 (2018): 233-238.

9. Jara, Antonio J., et al. "Semantic web of things: an analysis of the application semantics for the iot moving towards the iot convergence." *International Journal of Web and Grid Services*10.2-3 (2014): 244-272.

10. Chen, Baotong, et al. "Smart factory of industry 4.0: key technologies, application case, and challenges." *IEEE Access* 6 (2018): 6505-6519.

11. Wu, Zhenyu, et al. "Towards a semantic web of things: a hybrid semantic annotation, extraction, and reasoning framework for cyber-physical system." *Sensors* 17.2 (2017): 403.

12. McGinnes, Simon, and Evangelos Kapros. "Conceptual independence: A design principle for the construction of adaptive information systems." *Information Systems* 47 (2015): 33-50.

13. Bhérer, Louis et al., "Toward an Adaptive Application Builder: Two Communication Systems for an Ontology-Based Adaptive Information System Framework", International Journal On Advances in Intelligent Systems 9.1-9.2 (2016): 109-119.

14. Fielding, Roy T., and Richard N. Taylor. *Architectural styles and the design of network-based software architectures*. Vol. 7. Doctoral dissertation: University of California, Irvine, 2000.

15. Peinl, René. "Semantic web: State of the art and adoption in corporations." *KI-Künstliche Intelligenz* 30.2 (2016): 131-138.

16. Dadzie, Aba-Sah, and Emmanuel Pietriga. "Visualisation of linked data–reprise." *Semantic Web* 8.1 (2017): 1-21.

17. Liu, Shixia, et al. "A survey on information visualization: recent advances and challenges." *The Visual Computer* 30.12 (2014): 1373-1393.

18. Dadzie, Aba-Sah, and Matthew Rowe. "Approaches to visualising linked data: A survey." *Semantic Web* 2.2 (2011): 89-124.

19. Vega-Gorgojo, Guillermo, et al. "Visual query interfaces for semantic datasets: An evaluation study." *Journal of Web Semantics* 39 (2016): 81-96.

20. Soylu, Ahmet, and Martin Giese. "Qualifying ontology-based visual query formulation." *Flexible Query Answering Systems 2015*. Springer, Cham, 2016. 243-255.

21. Karger, David. "The pathetic fallacy of RDF." (2006).

22. Heim, Philipp, Thomas Ertl, and Jürgen Ziegler. "Facet graphs: Complex semantic querying made easy." *Extended Semantic Web Conference*. Springer, Berlin, Heidelberg, 2010.

23. March, James G. "Exploration and exploitation in organizational learning." *Organization science* 2.1 (1991): 71-87.

24. Medhe, Shrutika, and D. A. Phalke. "RDF data retrieval in structured format using aggregate function and keyword search in MashQL." (2013): 287-294.

25. Hoefler, Patrick, et al. "Linked Data Query Wizard: A Novel Interface for Accessing SPARQL Endpoints." *LDOW*. 2014.

26. L. Bhérer, L. Vouligny, M. Gaha, B. Redouane, and C. Desrosiers, "Ontology-based adaptive information system framework," in IARIA SEMAPRO 2015, The Ninth International Conference on Advances in Semantic Processing (2015) : 110–115.