Ontology-Based Adaptive Information System Framework

Louis Bhérer, Luc Vouligny
Mohamed Gaha, Billel Redouane

Institut de Recherche d'Hydro-Québec, IREQ
Varennes, Québec, Canada
Email: `Bherer.Louis2, Vouligny.Luc, Gaha.Mohamed, Redouane.Billel @ireq.ca`

Christian Desrosiers

École de Technologie Supérieure
Montréal, Québec, Canada
Email: `christian.desrosiers@etsmtl.ca`

***Abstract*—Software development does not usually end with the final release of the application. The software application have to be maintained throughout its useful lifetime in order to follow the users' needs. Most software applications are built around a rigid data models and modifications that must be performed on such data model impact the application, resulting in additional maintenance costs. The main focus of this work is to design and implement an ontology-based software framework for building information systems that can auto-adapt to evolving semantic data models. This framework has been used in the development of a client-server application as a proof of concept. This application can adapt dynamically to numerous changes that can be made in the model without recompilation of the client-side or the server-side of the application.**

Keywords–Adaptive Information System; Ontology; RDF; RDFS; OWL; Autonomic Computing.

I. Introduction

Most software applications are built around a rigid data model drawn from relational database (RDB) technologies. On one hand, RDB technologies are mature and performant when storing and accessing information. On the other hand, their data model are hard to change when modifications must be performed. The modification process of the software itself is rather time consuming as most of the changes in the data model will also require adjustments in the corresponding objects' model. The evolution usually requires a transitional program to transfer stored information to the new data model, the recompilation and the republishing of the application. Usually, when the application is on a client-server system in a large organization, all this work must be synchronized between different departments.

Ontologies can also be used to model information. They can be established and refined as new knowledge is acquired and needs evolve. Ontologies repository technologies such as triplestores can be used in software applications that can be built to take into account how the data model evolves. However, current programming languages, such as C, C#, Java, etc., usually require a compilation process in order to adapt to an evolving data model.

Staab et al. [1] "[...] recommend that the ontology engineer gathers changes to the ontology and initiates the switch-over to a new version of the ontology after thoroughly testing possible effects to the application[...]". We deduct that the ease of model modification in the ontology can be constrained by the applications' rigid development framework and resulting programs. Applications able to self-adapt to data models would certainly bring cost reductions on both development and maintenance processes.

The main focus of this work is to design and implement a framework for building an information system that can auto-adapt to evolving semantic data models. This framework has been used in the development of a client-server application as a proof of concept. This application can adapt dynamically to numerous changes that can be made in the model without recompilation of the client-side or the server-side of the application. The goal of this framework is to reduce the costs associated with application development, deployment and maintenance at Hydro-Québec, Québec's provincial utility that generates, transmits and distributes electricity. At IREQ, the research institute of Hydro-Québec, studies on the application of semantic technologies are currently underway as a mean to solve problems related to the increasing number of databases in the organization [2][3]. In addition, self-adapting technologies have already been applied with success [4].

The remainder of this article is organized as follows. The next section is a review of the previous works on system/framework with self-adaptive capacities. Section III presents the framework, its design and main functions, as well as the application built with it. Section IV presents the results of the project. Finally, Section V will cover the potential applications and advantages of this framework and future developments.

II. Related work

In 2001, IBM has proposed the Autonomic Computing initiative [5] with the objective to develop mechanisms that would allow systems and subsystems to self-adapt to unpredictable changes. Conferences, such as Software Engineering for Adaptive and Self-Managing Systems (SEAMS) [6] or Engineering of Autonomic and Autonomous Systems (EASe) [7], show that system and software self-adaptability is still an important research area, now scattered in a variety of subfields. Amongst them, one could include information system self-adaptability to an evolving data model.

As Dobson & al. stressed out in [8], the Autonomic Computing initiative did not achieve the promises announced in [9]. Many individual advances have brought some of those expected benefits, but there is no integrated solution resulting in an autonomous system. This is a task that some researchers have started working on, such as Bermejo-Alonso in [10] with her attempt to develop an ontology for the engineering of autonomous systems. The self-adaptability mechanisms of our framework could help in the development of self-aware or self-adjusting properties [11] leading to the development of autonomic components.

At Hydro-Québec, advances have been made in self-adapting applications with the Dynamic Information Modelling (MDI) development environment [4]. Some client-server applications built using this system have been put in production and are still in use today. Self-adaptation, even though it is only to the data model, have proven to be beneficial, especially when evolutionary prototyping is used as a development methodology [12]. In MDI, the proposed development library was not a client-server framework and was used as a private and closed semantic modeling system.

In [13], McGinnes and Kapros circumscribe the problem of non-adaptive applications as a conceptual dependence to the data model. They describe this dependence between the data model and the resulting application as an undesirable software coupling. The authors use the terms “Adaptive Information System” (AIS) for information systems that adapt to changes to the underlying data model. They conclude that most applications based on information systems used today are dependant on their domain model. Therefore, such systems must be maintained every time there is a modification on the data model and even the slightest change may result in costly and time consuming adaptations.

McGinnes and Kapros propose six principles to achieve conceptual independence over any data source (see Table 1). Using these principles, they show that it is possible to build an AIS based on an Extensible Markup Language (XML) mapping of a RDB data source [13]. Applying those principles to Resource Description Framework (RDF) based ontologies brings useful insights (see Table 1) on the use of those technologies in an AIS. One can argue that achieving conceptual independence using RDF-based technologies such as RDF, Resource Description Framework Schema (RDFS) and Web Ontology Language (OWL) seems more intuitive than using RDB data sources. RDF-based technologies have in fact many of the required properties inherently built in their design, thus reducing the complexity of achieving conceptual independence.

The proposed AIS framework based on semantic technology is presented in the next section.

III. Proposed AIS Framework

Our AIS has been conceptualised and developed as a three-tier client-server framework: a triplestore, a generic server and a web interface.

The triplestore is used to hold the knowledge bases constituted by a conceptual model and its individuals. In the proposed AIS, two knowledge bases are used: one for the domain of expertise and one for the presentation of the information. The triplestore used in this framework is Oracle 12c RDF semantic Graph Triplestore.

The server-tier is coded using a standard Java J2EE technology. It is built as a web service server offering different generic functions with a REST client-server interface. These services are implemented using the library JENA to process the requests written in the SPARQL query language.

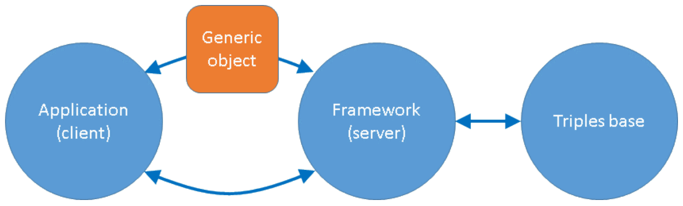


Figure 1. AIS framework.

The user interface is implemented in JavaScript with the Ext.js 4.2.2 library. It uses the REST interface to communicate with the server. Thus, it is independent to the server and could be coded using another technology.

We used the proposed framework to implement a decision support system application to be used at the Hydro-Québec research institute. The purpose of this application is to gather power transformer oil sampling data such as methanol and ethanol concentrations to monitor the health condition of the power transformers and provide suitable maintenance advice to the specialists. The application acts as a dashboard in which the users can add, update or delete entries and do simple searches. It also does automated calculations e.g., to calculate adjusted concentrations of some molecules depending on the temperature of the oil. The engineers will use the application to record their maintenance operations and measurements, to follow and compare the condition of the transformers and to test and refine parameters used in concentration adjustment equations.

The conceptual model of this application comprises six classes that will be used in the subsequent examples: PowerStation, PowerTransformer, Measurement, MaintenanceIntervention, ConversionParameter and PowerStationAndTransformerAssociation. Each of these classes has between two and twelve properties and comprise up to 7000 individuals. This application has been chosen to validate the framework since it requires a variety of functionalities that would be suitable for a wide range of applications.

In order to better understand the proposed AIS, Figure 1 presents a high-level view of the framework. At the initialization phase, the application requests the triplestore via a web service to show the initial presentation consisting of a tree view of the data model. The user uses this tree to select an individual of a class (e.g., a power transformer), represented by a leaf of the tree. When the user clicks on this leaf, the interface sends a request to the server through its web services. Upon reception of the request, the server dynamically gathers an undetermined number of classes, all of which have an association relation with the class of the selected individual. The server then gathers for each of these classes, the list of its properties and the list of its individuals related to the user’s selection. This information is transmitted to the client using a generic Java object and its corresponding JSON representation. This generic object is used to transfer the information to appear on the user’s interface and to request Create, Read, Update and Delete (CRUD) operations to the server.

A. *Application triplestore setting*

Most of the application data are stored in an enterprise RDB. A semantic meta-model (T-Box) has been designed to

TABLE I. CONCEPTUAL INDEPENDENCE PRINCIPLES AND APPLICATIONS.

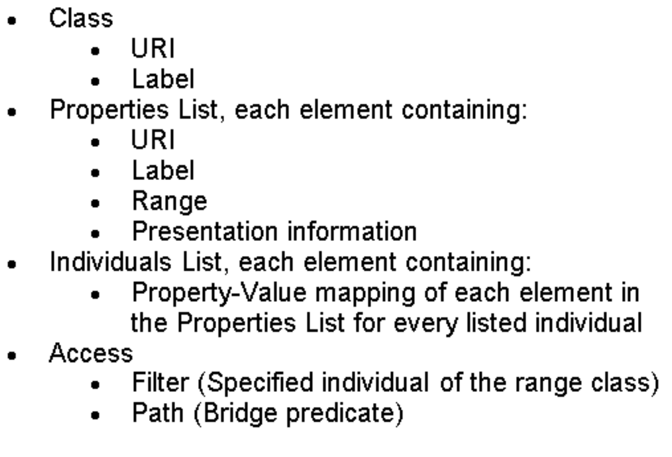
CONCEPTUAL INDEPENDENCE PRINCIPLES [13]	**APPLICATIONS OF THE CONCEPTUAL INDEPENDENCE PRINCIPLES WITH RDF-BASED TECHNOLOGIES**
1. Reusable functionality (structurally- appropriate behaviour): The AIS can support any conceptual model. Domain-dependent code and structures are avoided. Useful generic functionality is invoked at run time for each entity type. [13]	This principle applies similarly using a triplestore data source. Generics SPARQL requests will be obtained by exclusively hard-coding resources from the RDF, RDFS or OWL semantics, leaving the others resources soft-coded. The data model can be inspected at run time using generic SPARQL requests.
2. Known categories of data (semantically- appropriate behaviour): Each entity type is associated with one or more predefined generic categories. Category-specific functionality is invoked at run time for each entity type. [13]	All ontologies using RDFS or OWL languages contain *ipso facto* the same conceptual basis. The definition of those meta-entities are the semantics of RDF, RDFS and OWL. Employing those meta-entities as the most generic entities of the AIS allows the use of any RDF-based ontology. McGinnes and Kapros use archetypal categories taken from the field of psychology to classify entities according to the behaviours the AIS should adopt in their presence. This interesting idea will be considered later on in the development of this AIS, but is not currently essential.
3.Adaptive data management (schema evolution): The AIS can store and reconcile data with multiple definitions for each entity type (i.e.,multiple conceptual models), allowing the end user to make sense of the data. [13]	First, RDF technology uses what McGinnes and Kapros call soft-schemas: data models stored as data. Secondly, RDF technology allows individuals with different valued properties to coexist in the same class. Moreover, individuals can belong to more than one class. Axioms like *OWL:sameAs* or *OWL:equivalentClass* allow to reconcile data from distinctly described entities. Two previously distinct classes declared as equivalent will have, by inference, the same set of properties and then two individuals of this new class may have only different valued properties. Thus, this mechanism allows for reconciliation of data from different conceptual models. As the model evolves, data using different conceptual models remain available and is instantly accessible without any refactoring of the AIS.
4. Schema enforcement (domain and referential integrity) : Each item of stored data conforms to a particular entity type definition, which was enforced at the time of data entry (or last edit). [13]	In technologies such as OWL, domain integrity and referential integrity can be validated with reasoners. As for data types, literal data are usually associated with basic types upon entry in a semantic store.
5. Entity identification (entity integrity): The stored data relating to each entity are uniquely identified in a way which is invariant with respect to schema change. [13]	In the RDF technology, entity identification is provided by the URI mechanism, and is already invariant with respect to schema change.
6. Labelling (data management): The stored data relating to each entity are labelled such that the applicable conceptual models can be determined. [13]	Using the RDF technology, this principle would translate as: each individual needs to belong to a class. Then, is does not matter how much the class has change over time, because all of its individuals can have any number of valued or non-valued properties. However, human-readable labels are necessary to present the information to the users and it is mandatory to affect each entity with such labels.

model the required classes (PowerTransformer, PowerStation, etc.). Then, by using the D2RQ library, the data from the RDB have been converted into a RDF individuals graph (A-Box). The T-Box has been designed using RDFS semantics. It solely contains association relationships, and essentially describes the classes and the properties with their domain and range. Each class, property and individual have been labeled in order to be shown on the visual interface.

B. *Dynamic visualization of the semantic data*

Here are the main design elements for the dynamic visualization of the information.

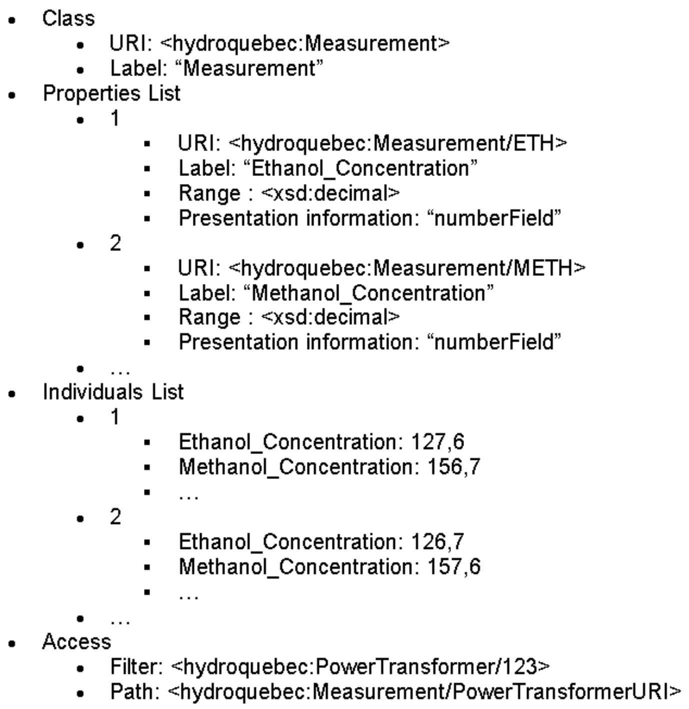
1) The generic object: A generic Java class (meta-class) was designed in order to allow dynamic recuperation of information from the semantic store. The resulting object is used to transfer information from the semantic store to the user's interface. A given object's instance is built from generic SPARQL requests using RDF and RDFS semantics. The object has fixed attributes used to hold information on the RDFS class, its properties and individuals. The generic object can also hold the path and filters used to select the individuals or the class itself. See Figure 2 for the definition of the object.



- Class
  - URI
  - Label
- Properties List, each element containing:
  - URI
  - Label
  - Range
  - Presentation information
- Individuals List, each element containing:
  - Property-Value mapping of each element in the Properties List for every listed individual
- Access
  - Filter (Specified individual of the range class)
  - Path (Bridge predicate)

Figure 2. Definition of the Java generic object.

Note here that each individual contains a property-value mapping for each property of the Properties List, and its corresponding value, if any. The access elements contain the



- Class
    - URI: <hydroquebec:Measurement>
    - Label: “Measurement”
- Properties List
    - 1
        - URI: <hydroquebec:Measurement/ETH>
        - Label: “Ethanol_Concentration”
        - Range : <xsd:decimal>
        - Presentation information: “numberField”
    - 2
        - URI: <hydroquebec:Measurement/METH>
        - Label: “Methanol_Concentration”
        - Range : <xsd:decimal>
        - Presentation information: “numberField”
    - …
- Individuals List
    - 1
        - Ethanol_Concentration: 127,6
        - Methanol_Concentration: 156,7
        - …
    - 2
        - Ethanol_Concentration: 126,7
        - Methanol_Concentration: 157,6
        - …
    - …
- Access
    - Filter: <hydroquebec:PowerTransformer/123>
    - Path: <hydroquebec:Measurement/PowerTransformerURI>

Figure 3. Example of a Java generic object.

path in the graph to get to the class (i.e., the property linking the individuals of the two classes) and the filter (i.e., an individual of the range class) used to select the individuals of the domain class. The term “Bridge predicate” will be used to refer to the property linking the domain class and the range class (i.e., the path) (See Figure 4).

In the developed application, selecting a power transformer in the tree will result in a request to find individuals linked to it from all classes having a property whose range is the Transformer class, i.e., individuals from the domain classes of the Transformer class. For each class found, a generic object will be created.

In order to better understand how generic objects are created, please refer to the example given in Figure 3. In this example, the user has selected the power transformer numbered 123. The framework then requested the model and found three classes having an associative relation with the PowerTransformer class: Measurement, MaintenanceIntervention and PowerStationAndTransformerAssociation. Those three classes are going to be fetched but this example presents only the Measurement class case. Its URI and label have been first retrieved, followed by the list of its properties and the list of its individuals. This second list contains a mapping for each individual, between every properties of the property list and its value for this individual, if any.

In the example in Figure 3, the filter is the specified individual of the range class, i.e., the power transformer numbered 123. It is considered a filter because it reduces the number of individuals retrieved. Here, the path is simply the Bridge predicate between the range and the domain classes. Further development should lead to the creation of more filter and path options, as well as sequences and aggregations of these options.

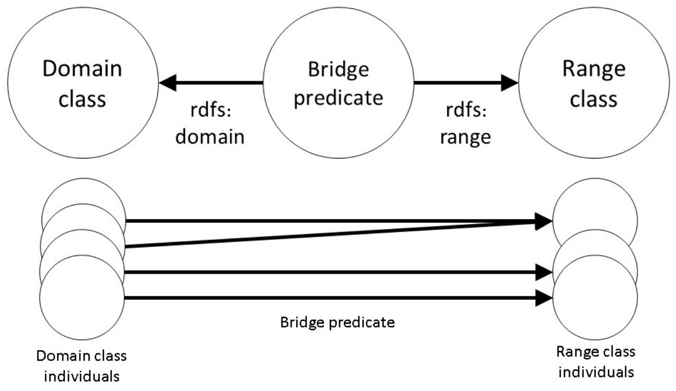


Figure 4. Graph representation of the range and domain classes in an associative relationship.

In our AIS, every time a power transformer is selected in the tree, the model is inspected dynamically to find all the domain classes of the PowerTransformer class and all their individuals linked to the selected power transformer. Hence, if a new domain class is added, the application will automatically present it to the users.

Due to the genericity of the functions, the changes made on the data model are immediately available to all the AIS users. From then on, every request will get individuals and classes from the new model, without any need to recompile the client nor the server. This is due to the fact that being written to adapt to any model, a request can then be used in run time to inspect the actual version of the model.

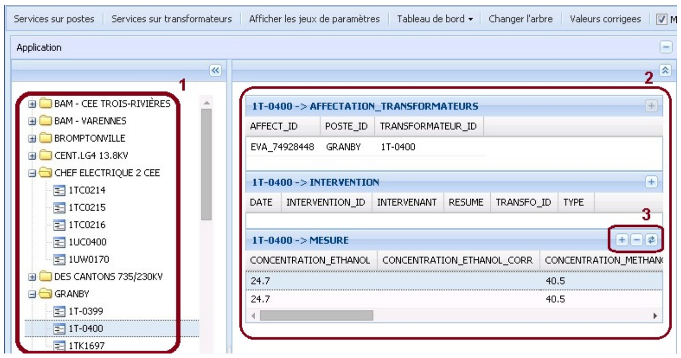
2) Visual representation: The application uses a tree to show the user a specific portion of the semantic graph (see Figure 5). In our case, the tree first shows all the power stations as folders that can be expanded to see the power transformers they contain.

When the user selects a node (e.g., the power transformer 123), the client user interface sends a request to the AIS server, using a generic process, to dynamically gather the domain classes (e.g., the Measurement class) in relation with the range class (e.g., the PowerTransformer class). For each of these classes, the properties will first be found, and then, all the individuals of the domain classes linked with the user selected individual will be retrieved. As a result, a list of generic Java objects will be generated where each object corresponds to a domain class.

These Java objects are then automatically converted to JSON, using the Jackson library [14] and sent to the user interface. The user interface will produce a bidimensional matrix for every class in the list (see Figure 5). These matrices show the information to the user using human-readable labels. The user can then request for CRUD operations on individuals represented in the matrices (see Figure 5).

The CRUD operations are programmed to retrieve the Java generic object and delete all the unselected individuals, so to keep only the selected individual. This individual is then modified according to the user’s needs and the resulting Java object is returned to the server.

In the current state of the framework implementation, if changes are made in the T-Box, either by modifying the



BAM - CEE TROIS-RIVIÈRES
BAM - VARENNES
BROMPTONVILLE
CENT.LG4 13.8KV
CHEF ELECTRIQUE 2 CEE
1TC0214
1TC0215
1TC0216
1UC0400
1UW0170
DES CANTONS 735/230KV
GRANBY
1T-0399
1T-0400
1TK1697

1T-0400 -> AFFECTATION_TRANSFORMATEURS

AFFECT_ID	POSTE_ID	TRANSFORMATEUR_ID
EVA_74928448	GRANBY	1T-0400

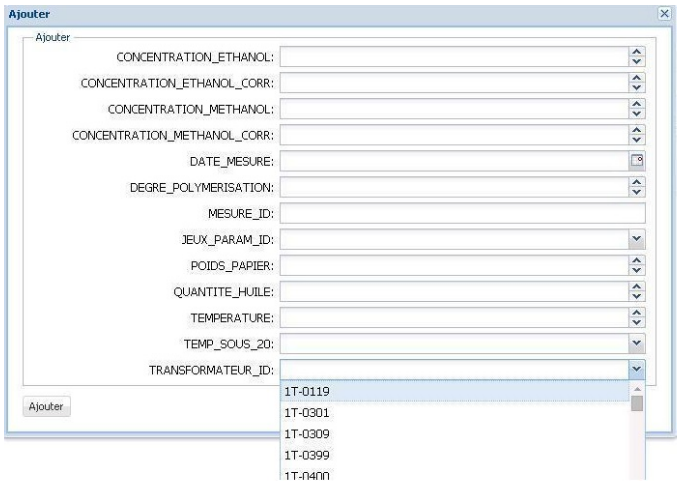
1T-0400 -> INTERVENTION

DATE	INTERVENTION_ID	INTERVENANT	RESUME	TRANSFO_ID	TYPE

1T-0400 -> MESURE

CONCENTRATION_ETHANOL	CONCENTRATION_ETHANOL_CORR	CONCENTRATION_METHAN
24.7		40.5
24.7		40.5

Figure 5. The tree view (1), matrices (2) and the buttons to request for CRUD operations (3).



CONCENTRATION_ETHANOL:	
CONCENTRATION_ETHANOL_CORR:	
CONCENTRATION_METHANOL:	
CONCENTRATION_METHANOL_CORR:	
DATE_MESURE:	
DEGRE_POLYMERISATION:	
MESURE_ID:	
JEUX_PARAM_ID:	
POIDS_PAPIER:	
QUANTITE_HUILE:	
TEMPERATURE:	
TEMP_SOUS_20:	
TRANSFORMATEUR_ID:	

1T-0119
1T-0301
1T-0309
1T-0399
1T-0400

Figure 6. Individual CRUD form example.

properties of some domain classes or by adding a new domain class related to the class of the tree leaves, the users will instantly begin to navigate in the new conceptual model. Other changes are not yet possible.

The main presentation tree does not grant access to every class in the semantic graph. Therefore, the user interface has been given other access points from which the user can request directly for those previously inaccessible classes. The system uses a similar generic function to request this information except that it retrieves the class itself and all its individuals instead of using the previously presented domain classes mechanism. The same generic Java object is used, but does not have any access information. As the generic Java object is used, the same CRUD operations can still be performed on individuals.

3) The CRUD services: For the time being, the framework allows CRUD services on the individuals only, not on the classes and properties. Other means are used to edit the conceptual model. Further work will be made to allow modelization of the T-Box from the user interface. The CRUD services on the A-Box are done on the client-side using forms showing the properties of the class and their value for the selected individuals, if any (see Figure 6). These forms are created from the properties listed in the generic Java object.

In order to help the user and validate the input, a presentation knowledge base comprising the different presentation options has been established. This information is associated with every property of the domain knowledge base and is passed on by the Java generic object. It indicates how to build every entry fields of the forms. Those forms are constructed dynamically, adapting the user's interaction options on the values of properties according to the presentation knowledge base information.

In further developments, mechanisms will be designed to automatically link the domain ontology properties to the presentation ontology individuals. Some ontologies contain semantics, such as Enumeration, Sequence, or Bag, that can be used to predict the correct entry field's type for a certain property. Enumeration, for example, can be represented as a list of individuals from which the user will have to choose. In general, the range of a property is a good indicator of the required entry field's type. Finally, functions will be implemented to allow the user to change the type of the entry field in run time.

In the current state of the framework, four types of entry fields are implemented: numerical fields, text fields, list fields and date fields. Upon expansion, the list field requests for a service that finds all the existing values associated to this property. For the fields used to update literals, the range type of the property is used for validation. Cardinalities are present in the presentation knowledge base so the forms can indicates to the user the required fields, if any.

4) The graphics: Graphic classes and related properties have been added to the presentation knowledge base to represent graphic views such as histograms or clouds of points. Graphic properties are used to specify the association between the graphic elements such as the x-axis data, y-axis data, labelings, etc. The axis are linked to domain ontology properties. When these domain ontology properties are present in generic objects, the user interface could detect them and create a list of available graphics.

IV. Results

The framework has been used to create a client-server decision-support application. Thanks to the generic services of the AIS framework, one can modify the classes and properties in the conceptual model directly in the triplestore without affecting the application. The user interface will adjust its presentation automatically according to the latest update of the conceptual model, since the request interrogates the semantic graph dynamically. The proposed framework allows for all CRUD operations to be performed on individuals. Moreover, the framework will query the conceptual model in the semantic graph for each request, which is different to a standard application where the conceptual model is taken into consideration only at compilation time. The resulting application is ready to be put in production. Once in production, because it will be able to automatically adapt to conceptual model changes, it should easily evolve as the framework is extended.

The main limitation of this framework is how it explores the model at each request. While for now it only retrieved individuals from classes that are one associative relation away from a desired individual, further work is needed to find ways of expanding this exploration. The implementation of this mechanism will be crucial for the framework to be effective in large scale ontologies.

Tests still need to be run to determine performance differences between such an application and a non-dynamic one, and to observe the scaling potential. The resulting application from the proposed framework is not expected to be as performant as a similar application developed from a more conventional framework, but the difference in performance has yet to be established. Then, it will be possible to evaluate how much the cost reductions incurred during the development and the maintenance processes of AIS outweighs their performance aspect on the long run.

While implementing this proof of concept, we learned that many of the needed properties to achieve conceptual independence are inherent to the RDF technology. Exclusively hard-coding resources from the RDF, RDFS and OWL semantics in all the SPARQL requests and leaving all other resources soft-coded are necessary conditions to obtain this conceptual independence. Because the semantics of these three languages (RDF, RDFS and OWL) are shared across all RDF-based ontologies, they form a common conceptual basis to all the domains they can represent. Limiting the conceptual dependences to their semantics, the applications built can use any such ontology, regardless of its knowledge domain.

V. CONCLUSION AND FUTURE WORK

As hypothesised, an AIS based on a triplestore seems easier to implement than an AIS using XML to dynamize functions on a RDB. Many artifices have to be considered when building an AIS from a RDB which are not required with semantic technologies, as described in Table 1. The use of a library to map the RDB into a triplestore appears judicious to easily and quickly gather the conceptual independence needed in an AIS.

With the use of a RDF representation to store the information, generic SPARQL requests that can search any semantic graph for both conceptual knowledge and individual information are easily devised. This leads the AIS to be able to adapt to the evolution of the conceptual model and to be used for different domains of application. The framework could also be used with evolutionary prototyping application development. At Hydro-Québec, other large scale client-server applications have already been successfully developed using evolutionary prototyping, highlighting the benefits of such technologies compared to standard development processes [12].

The construction of an application editor able to use the framework for developing new auto-adaptive applications seems to be the next logical step. Using the framework to build new applications will further test the approach and allow to complete the presentation knowledge base. In doing so, new functions will be developed leading eventually to a complete AIS. Ideally, the AIS should be able to take advantage of all the RDF, RDFS and OWL semantics.

The current application uses only RDFS semantics; adding OWL capabilities will allow for the use of inference reasoners. In the current release, only the individuals of the semantic domain can be edited by the user through forms. Editing possibilities on the meta-model will be authorized in the next iterations.

The framework and the application are the proof that an AIS can work easily and efficiently by capitalizing on the RDF technology and its inherent properties. Such systems can be useful in fast-evolving knowledge domains. They inscribe themselves well in the AGILE development philosophy, allowing the model data to evolve freely at each iteration. Those considerations allow to think that AIS and self-adapting applications could bring substantial cost reductions in application development and maintenance in the coming years.

ACKNOWLEDGMENT

The authors would like to thank Jérôme Côté for his insightful help.

REFERENCES

[1] S. Staab, R. Studer, H.-P. Schnurr, and Y. Sure, “Knowledge processes and ontologies,” IEEE Intelligent Systems, vol. 16, no. 1, Jan. 2001, pp. 26–34.

[2] A. Zinflou, M. Gaha, A. Bouffard, L. Vouligny, C. Langheit, and M. Viau, “Application of an ontology-based and rule-based model in electric power utilities,” in 2013 IEEE Seventh International Conference on Semantic Computing, Irvine, CA, USA, September 16-18, 2013, 2013, pp. 405–411.

[3] M. Gaha, A. Zinflou, C. Langheit, A. Bouffard, M. Viau, and L. Vouligny, “An ontology-based reasoning approach for electric power utilities,” in Web Reasoning and Rule Systems - 7th International Conference, RR 2013, Mannheim, Germany, July 27-29, 2013. Proceedings, 2013, pp. 95–108.

[4] L. Vouligny and J.-M. Robert, “Online help system design based on the situated action theory,” in Proceedings of the 2005 Latin American Conference on Human-computer Interaction, ser. CLIHC ’05. New York, NY, USA: ACM, 2005, pp. 64–75.

[5] P. Horn, “Autonomic Computing: IBM’s Perspective on the State of Information Technology,” Tech. Rep., 2001.

[6] T. Vogel. Software engineering for self-adaptive systems. [retreived: 06, 2015]. [Online]. Available: https://www.hpi.uni-potsdam.de/giese/public/selfadapt/ (2015)

[7] I. T. C. on Software Engineering. Ieee ease 2014. [retreived: 06, 2015]. [Online]. Available: http://tab.computer.org/aas/ease/2014/index.html (2014)

[8] S. Dobson, R. Sterritt, P. Nixon, and M. Hinchey, “Fulfilling the vision of autonomic computing,” Computer, vol. 43, no. 1, 2010, pp. 35–41.

[9] J. O. Kephart and D. M. Chess, “The vision of autonomic computing,” Computer, vol. 36, no. 1, Jan. 2003, pp. 41–50.

[10] J. Bermejo-Alonso, R. Sanz, M. Rodríguez, and C. Hernández, “Ontology-based engineering of autonomous systems,” in Proceedings of the 2010 Sixth International Conference on Autonomic and Autonomous Systems, ser. ICAS ’10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 47–51.

[11] R. Sterritt and M. Hinchey, “Spaace iv: Self-properties for an autonomous computing environment; part iv a newish hope,” in Engineering of Autonomic and Autonomous Systems (EASe), 2010 Seventh IEEE International Conference and Workshops on, March 2010, pp. 119–125.

[12] L. Vouligny, C. Hudon, and D. N. Nguyen, “Design of mida, a web-based diagnostic application for hydroelectric generators.” in COMPSAC (2), S. I. Ahamed, E. Bertino, C. K. Chang, V. Getov, L. L. 0001, H. Ming, and R. Subramanyan, Eds. IEEE Computer Society, 2009, pp. 166–171.

[13] S. McGinnes and E. Kapros, “Conceptual independence: A design principle for the construction of adaptive information systems,” Inf. Syst., vol. 47, 2015, pp. 33–50.

[14] Cowtowncoder. Jackson. [retreived: 06, 2015]. [Online]. Available: http://jackson.codehaus.org/ (2015)