# Toward an Adaptive Application Builder: Two Communication Systems for an Ontology-Based Adaptive Information System Framework

Louis Bhérer
Luc Vouligny, Mohamed Gaha

Institut de recherche d'Hydro-Québec, IREQ
Varennes, Québec, Canada
Email: bhererlouis@gmail.com
Vouligny.Luc@ireq.ca, Gaha.Mohamed@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 must be maintained throughout its lifetime to keep in step with the user's needs. Most software applications are built around a rigid data model, which whenever modified will have an impact on the application, resulting in additional maintenance costs. A way to mitigate this problem would be to have an ontology-based software framework for building information systems that can auto-adapt to an evolving data model. Such a framework has been built and 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 data model without recompiling the client side or server side of the application. Two communication systems between the client and server have been tested to compare their performance, code length and capabilities. In order for this framework to be efficiently used for the development of applications, it must be combined with other components to form an adaptive application builder, whose design is discussed with regard to the ontology-driven architecture paradigm.**

***Keywords–Adaptive Information System; Ontology; RDF; RDFS; OWL; Ontology-Driven Architecture; Model-Driven Engineering, Autonomic Computing.***

## I. INTRODUCTION

This article is an extended version of an earlier paper that described an ontology-based framework for building applications capable of adapting to changes in the data model. Here, it will be explained how this framework can be part of a more complex entity: an adaptive application builder (AAB). In addition, two communication systems between client and server are compared [1].

Many of today's software applications are developed around a rigid data model drawn from relational database (RDB) technologies. Though RDB technologies are mature and perform well when storing and accessing data, their data models are hard to change when modifications must be made. Modification of the software itself is rather time-consuming as most of the changes to the data model also require adjustments to the corresponding objects' model. Migration usually requires a transitional program to transfer stored information to the new data model, recompile and republish the application. Usually, when the application is on a client-server system in a large organization, all this work must be synchronized between departments, adding to the overall refactoring effort.

When ontologies are used to model information, they can be established and refined as new knowledge is acquired and as needs evolve. However, the ease with which models are modified within the ontology can be constrained by the applications' rigid development framework and resulting programs. Staab *et al.* recommend gathering all the modifications made to an ontology in order to test all possible consequences to the applications before deploying a new version [2]. We conclude that developing applications able to self-adapt to data models would reduce both development and maintenance costs. This paper shows how ontology repository technologies such as triplestores, i.e., database management systems (DBMS) for data modeled using the resource description framework (RDF) [3], can be used in applications built to take into account how the data model evolves. Moreover, such applications do not require a compilation process in order to adapt to an evolving data model, contrary to most current applications.

The main focus of this work is the design and implementation of a framework that allows for a fast and easy building of information systems that can auto-adapt to evolving data models. This framework has been used in the development of a client-server application as a proof of concept. The application adapts dynamically to numerous changes that can be made in the model without recompile of the client side or 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, the utility that generates, transmits and distributes electricity in the province of Québec. At IREQ, Hydro-Québec's research institute, studies on the application of semantic technologies are currently underway as a means to solve problems related to the increasing number of databases within the organization [4][5]. In addition, self-adapting technologies have already been applied successfully [6]. Since the adaptive properties of applications hinges on the communication system between the server side and the client side, two communication approaches have been tested and compared within the framework developed. The main contribution of this paper is the development of this framework as a new approach to generating an information system that auto-adapts to its data model.

In order to give the framework more complex adaptive capabilities and facilitate its implementation into future company applications, an adaptive application builder (AAB) is being

developed. Since this AAB influences the development of the framework, its main principles will be discussed. As semantic technologies enable domain-independent meta-modeling of information systems, model-driven engineering (MDE) seems a natural approach to guide the development of an AAB.

Section II of this paper reviews earlier work in two areas: MDE and ontology-driven architecture (ODA), and then systems, frameworks and user interfaces (UI) with self-adaptive capabilities. Section III presents the framework, its design and main functions, an application built with it and an alternative communication system for the framework. Section IV presents the results, including the comparison between the two communication systems. Section V discusses the benefits of the proposed framework and future developments.

## II. Related work

This section first examines how MDE and semantic technologies relate to one other and what has been done to bridge the two worlds, and second, reviews works on auto-adaptiveness, or more generally on the self-* properties of applications and information systems.

### A. Model-driven engineering and ontology-driven architecture

Since the 1990s, MDE methodologies have focused on enabling software development from models. In order to maximize productivity, the model of the software application would be sufficient for programs to automatically generate the software itself. Though there are some success stories in MDE, it has not become a universal approach since adopting it remains complex and time consuming. Even so, researchers and companies are still devoting much attention to MDE because of its great promises [7]. In [8], Douglas C. Schmidt described MDE as a promising approach to shield developers from platform complexity, just as early programming languages protected programmers from the complexity of machine code. Schmidt stated that third-generation languages failed to alleviate this complexity due to their own complexity and to the rapid evolution and proliferation of platforms.

Ontologies have also been a popular research subject in the recent years, due primarily to their interoperability capabilities, which could facilitate the advent of the semantic web. Since an ontology is the “description of the concepts and the relationships that can formally exist for an agent or community of agent” [9], it is understandable why researchers have tried to use them with MDE.

The object management group (OMG), proponents of the model-driven architecture (MDA) [10], states that the goals of their approach include application re-use, complexity reduction, cross-platform interoperability, domain specificity and platform independence. Since these are also objectives of semantic technologies, synergy may presumably be achievable by combining the two paradigms. The World Wide Web Consortium (W3C), providers of the foundation of the semantic web, suggests that even if MDA is a good framework for software development, it could be improved by the use of semantic web technologies to disambiguate domain vocabularies, validate model consistency and increase the expressivity of the constraints representation. By thus augmenting the OMG methodology stack, ontologies could lead to the rise of ODA [11].

Pan *et al.* propose using ontologies and MDA together to reap the best of both worlds [12]. Their approach is to build bridges between the ModelWare and OntologyWare technical spaces. Both technical spaces are constructed on different layers, from metalanguages (M3), languages (M2) and models (M1) to running instances (M0). By bridging each of these layers, the ontologies’ capabilities would be enabled in an MDA approach during software development. Ontologies should be integrated with model-driven software development in order to validate the consistency of models, guide software developers and causally connect specifications during the development process [12].

In what seems like a much simpler and straightforward way to bring those two worlds together, Martins Zviedris *et al.* describe how they automatically build ontology-based information systems [13]. Following the Sowa’s principle that every software system has an ontology, implicit or explicit [14], the authors believe it possible to develop a universal platform-independent meta-model using an MDA approach.

They do so by first developing an ontology of web applications and then instantiating this ontology each time they wish to build a new information system. An engine they built is then used to ”understand” the instance of a particular application and automatically generate its code. The result is a hard-coded non-adaptive application with a one-to-one mapping of the domain ontology classes and properties into JAVA classes. The resulting application can be easily rebuilt if any change is made in the data model but must be recompiled to use.

The main difference between the work of Pan *et al.* and of Zviedris *et al.* is that the former are trying to bring the capabilities of ontologies into the MDA world, while the latter are focusing on bringing MDA learning into semantic standards. Although the approach of Pan *et al.* to bridging all the standards enables the use of many tools already developed with both technologies, it is far more complex to implement. By ignoring OMG standards, Zviedris *et al.* more easily bring MDA learning into the semantic realm, but at the cost of coding their own MDA tools, i.e., their web application builder and their web application runtime engine. In our view, both teams are working to achieve ODA in opposite ways. Our ODA approach is similar to Zviedris, enhancing upon it by incorporating auto-adaptive capabilities.

### B. Self-* properties

In 2001, IBM proposed the Autonomic Computing Initiative [15] 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) [16] or Engineering of Autonomic and Autonomous Systems (EASe) [17] show that system and software self-adaptability is still an important research area, now divided into a variety of subfields. Amongst them, one could include information system self-adaptability to an evolving data model.

As Dobson *et al.* pointed out [18], the Autonomic Computing Initiative did not fulfill the promises announced [19]. Though many individual advancements have yielded some of the expected benefits, there is still no integrated solution resulting in an autonomous system. This is a task being undertaken by some researchers, such as Bermejo-Alonso who is attempting to develop an ontology for the engineering of

autonomous systems [20]. All those individual advancements fall into the category of "self-* properties", i.e., ways for systems to automatically maintain themselves throughout different scenarios [21].

The self-adaptability mechanisms of the framework we proposed could help develop self-aware or self-adjusting properties [22] leading to the development of autonomic components. Within the hierarchies of self-* properties of both Salehie and Tahvildari [23] and Berns and Ghosh [21], our framework would be classified as a set of self-configuring properties. Once the framework is integrated into an AAB, some self-optimizing properties are likely to emerge.

As the framework's ultimate goal is semi-automatic development of completely adaptive applications, it is important to look at current adaptive user interfaces (UIs), and systems to study their strengths and weaknesses. Recently, Akiki *et al.* [24] presented an overview of the most important adaptive UIs, evaluating and classifying them. They divided the design of adaptive UIs into two approaches: (1) MDE and (2) window managers and widget toolkits. They then proposed seven criteria to evaluate the two approaches and concluded that the first has more advantages than the second. They next established 21 criteria to evaluate adaptive model-driven development systems, specifying whether each criterion applies to architectures, techniques or development tools. Lastly, they evaluated many UIs adapting to their context of use. This set of criteria provides a valuable starting point and checklist for devising a new adaptive UI.

At Hydro-Québec, progress has been made in self-adapting applications with the dynamic information modelling (DIM) development environment [6]. Some client-server applications built using this system have been put into production and are still in use today. Self-adaptation, even though it is only to the data model, has proven to be beneficial, especially when evolutionary prototyping is used as a development methodology [25]. In DIM, the proposed development library was not a client-server framework and was used as a private, closed semantic modeling system. Those benefits were an incentive to continue following the evolutionary prototyping approach, as does the framework and even more so the AAB. The use of standards like RDF makes it possible to continue developing this research field in concert with other researchers world-wide.

McGinnes and Kapros circumscribe the problem of non-adaptive applications as a conceptual dependence upon the data model [26]. They describe this dependence between the data model and the resulting application as undesirable software coupling. The authors use the term "adaptive information system" (AIS) for an information system that adapts to changes made to the underlying data model. They conclude that most applications based on information systems in use today are dependent on their domain model. Such systems must thus be maintained every time the data model is changed, even the slightest change potentially resulting in costly, time-consuming adaptation.

McGinnes and Kapros propose six principles to achieve conceptual independence over any data source (see Table I). Using those principles, they show that it is possible to build an AIS based on an Extensible Markup Language (XML) mapping of an RDB data source [26]. Applying those principles to ontologies based on RDF brings useful insights (see Table I) on the use of those technologies in an AIS. Achieving conceptual independence using RDF-based technologies such as Resource Description Framework Schema (RDFS) and Web Ontology Language (OWL) is arguably more intuitive than using RDB data sources. RDF-based technologies actually have many of the required properties inherently built into their design, thus reducing the complexity of achieving conceptual independence. In Table I, principles 3, 5 and 6 may be considered inherent to this technology. Use of the other principles is discussed in Section III.

The proposed ontology-based adaptive information system framework (OBAISF) is presented in the next section. The applications built with OBAISF are conceptually independent from their data model, like McGinnes and Kapros but with independence achieved using semantic technologies.

## III. Proposed Ontology-Based Adaptive Information System Framework

As stated previously, our current goal is to develop an AAB that will use an ontology-based adaptive information system framework (OBAISF) to construct auto-adaptive applications. The AAB is the combination of an ontology browser (OB), a SPARQL query builder (SQB) and an OBAISF. SPARQL stands for "SPARQL Protocol and RDF Query Language" [27]. The OB and SQB designs will be the subject of a future paper. In the coming months, the OB will be enhanced with an editing module supporting the instantiation of a web application ontology (WAO), in line with the work of Zviedris *et al.* [13] presented in Section II. With this editing module, web application components can be added to a particular instantiation of the WAO and customized through OB and SQB functionalities.

### A. Proposed adaptive application builder

The OB has been developed to navigate any OWL ontology. It first presents all the classes of an OWL graph in a tree. Then, upon selection, a box shows all the properties of the selected object and those of its superclasses. From then on, the user can navigate the ontology using object-type properties, which open new boxes (Figure 1). In the AAB, the OB will be used to choose classes and properties of an ontology and link them to application components. For example, two classes linked by an object-type property could be selected so their instances become the branches and leaves of a data tree component.

The SQB is built on top of the OB and uses a recursive JAVA engine to translate a user-originated visual query into a proper SPARQL query. The SQB supports querying data-type properties and applying filters to them (Figure 1). In the AAB, the SQB will be used to build views on the semantic data. This way, the end user of an application can be presented with any view that could be made from the ontology.

All this is made possible by the semantic properties in Table I, which enable the creation of the OBAISF. The framework is basically a set of generic functions that dynamically query the ontology before querying a set of its instances. Those generic functions were first developed to be compatible with RDFS ontologies but are now being updated to also suit OWL ontologies. The remainder of this section will present an AIS built with an OBAISF for a decision support application.

TABLE I. CONCEPTUAL INDEPENDENCE PRINCIPLES AND APPLICATIONS.

| CONCEPTUAL INDEPENDENCE PRINCIPLES [26] | APPLICATIONS OF THE PRINCIPLES WITH RDF-BASED TECHNOLOGIES |
|---|---|
| 1. Reusable functionality (structurally- appropriate behavior): 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. | This principle applies similarly using a triplestore data source. Generic SPARQL requests will be obtained by exclusively hard-coding resources from the RDF, RDFS or OWL semantics, leaving other resources soft-coded. The data model can be inspected at run time using generic SPARQL requests. |
| 2. Known categories of data (semantically- appropriate behavior): Each entity type is associated with one or more predefined generic categories. Category-specific functionality is invoked at run time for each entity type. | All ontologies using RDFS or OWL languages contain *ipso facto* the same conceptual basis. The definition of those meta-entities is the semantics of RDF, RDFS and OWL. By employing those meta-entities as the most generic entities of the AIS, any RDF-based ontology can be used. 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 yet 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. | Firstly, 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* make it possible 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. This mechanism thus supports reconciliation of data from different conceptual models. As the model evolves, data using different conceptual models remains 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). | In technologies such as OWL, domain integrity and referential integrity can be validated with reasoners. As for data types, literal data is usually associated with basic types upon entry in a semantic store. |
| 5. Entity identification (entity integrity): The stored data relating to each entity is uniquely identified in a way that is invariant with respect to schema change. | In 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 is labelled such that the applicable conceptual models can be determined. | Using RDF technology, this principle means that every individual must belong to a class. It then does not matter how much the class has changed over time since all individuals in it 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 assign such labels to each entity. |

### *B. Proposed adaptive information system*

This AIS was developed as a three-tier client-server system: a triplestore, a generic server and a web interface.

The triplestore is used to store 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 information. The triplestore used in this framework is an Oracle 12c RDF Semantic Graph.

The server-tier is coded using a standard JAVA Enterprise Edition technology. It is built as a web service offering different generic functions with a REST client-server interface. These services are implemented using the Jena library [28] to process requests written in the SPARQL query language.

The user interface is implemented in JavaScript with the Ext.js 4.2.2 library [29]. It uses the REST interface to communicate with the server. It is thus independent from the server and could be coded using another technology.

We used our framework to implement a decision support application to be used at IREQ. The purpose of the application is to gather power transformer oil sampling data, such as methanol and ethanol concentrations, to monitor power transformer health and provide suitable maintenance advice to specialists. The application acts as a dashboard, within which the users can add, update or delete entries, and do simple searches. It also perform automated calculations, e.g., adjusting concentrations of specific molecules depending on the oil temperature. Engineers use the application to record maintenance operations and measurements, track and compare the health of transformers, and test and refine parameters used in concentration adjustment equations.

The conceptual model of this application has six classes that will be used in the subsequent examples:

1) PowerStation,
2) PowerTransformer,
3) Measurement,
4) MaintenanceIntervention,
5) ConversionParameter, and
6) PowerStationAndTransformerAssociation.

Each of these classes has between two and twelve properties and comprises up to 7,000 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 addition to having a small and simple ontology, easy to build and test at this stage of the project.

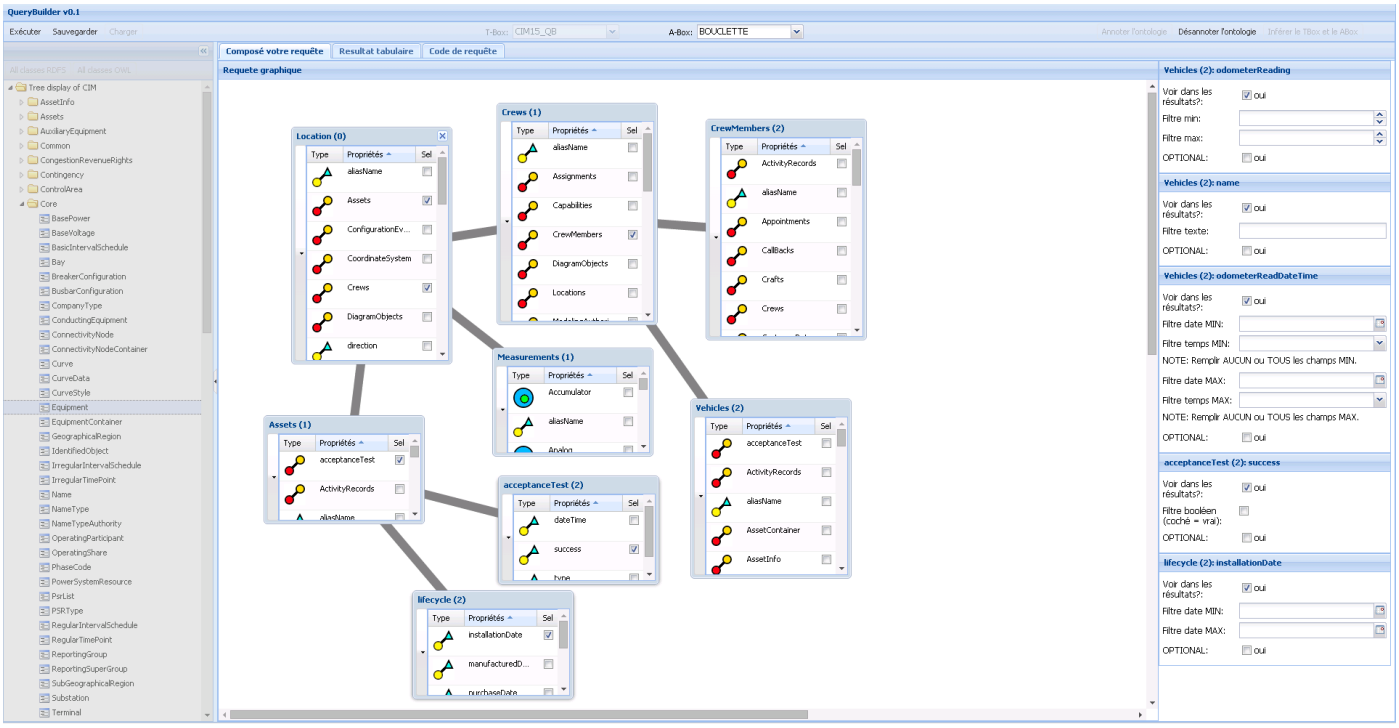


Figure 1. Ontology Browser and Sparql Query Builder overview: Center - Ontology Browser for navigating classes. Right - Filters on datatype properties to be used in the Sparql Query Builder.

*1) Application triplestore setting:* Most of this transformer monitoring application's data is stored in a RDB. A semantic meta-model (T-Box) has been designed to model the required classes (PowerTransformer, PowerStation, etc.). Then, by using the D2RQ library [30], the data from the RDB has been converted into an RDF graph of individuals (A-Box). The T-Box has been designed using RDFS semantics. It contains solely association relationships and essentially describes the classes and the properties with their domain and range. Each class, property and individual has been labeled in order to be displayed on the visual interface.

In order to better understand this application, Figure 2 presents a high-level view of its architecture. At the initialization phase, the application requests the triplestore via a web service to show a tree view of the data model. The user can view an individual of a class (e.g., a power transformer) by selecting a leaf in the tree. When the user clicks on this leaf, the interface sends a request to the server through its web service. Upon receipt of the request, the server dynamically gathers a number of classes determined by the model, all of which have an association relationship with the class of the selected individual. For each of these classes, the server will then gather the list of its properties and the list of its individuals related to the user's selection. This information can be transferred to the client using two systems: (1) a generic JAVA object and its corresponding JSON representation or (2) a generic triple model and its corresponding JSON-LD representation. A triple is "the fundamental RDF structure" [31] consisting of a subject, a predicate and an object. The W3C describes JSON-LD as "a JSON-based format to serialize Linked Data [whose] syntax is designed to easily integrate into deployed systems that already use JSON" [32]. As both JSON and JSON-LD

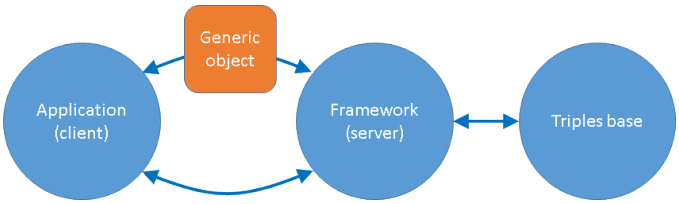


Figure 2. Adaptive information system framework.

share a similar data structure, they can both be processed by most web APIs or frameworks, with the difference that JSON-LD must be processed by a library beforehand. Those generic objects or models contain all the information to display on the UI and to request for further operations to the server, as Create, Read, Update, and Delete (CRUD) operations.

The next section will describe both data communication systems (using a generic JAVA object or using a generic triple model).

*2) Two systems for dynamic visualization of the semantic data:* Here are the main design elements for both systems for dynamic visualization of the information.

*a) System 1 - Generic object system:* In this system, a generic JAVA class (meta-class) was designed to support dynamic gathering of information from the semantic store. The resulting object is used to transfer information from the semantic store to the UI. 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 about the RDFS class, its properties and individuals. It also holds the path and filters used to select the individuals or the class

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

Figure 3. Definition of the JAVA generic object.

itself. See Figure 3 for the definition of the object.

Note here that each individual contains a property-value mapping for each property in the list of properties and its corresponding value, if any. The access elements contain the path in the graph to go to the class (i.e., the property linking the individuals of the two classes) and the filter (i.e., an individual in the range class) used to select the individuals in the domain class. The term "Bridge predicate" will be used to refer to the property linking the domain class and range class, i.e., the path (see Figure 5).

In the application developed, 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 is created.

The example in Figure 4 helps to better understand how generic objects are created. In this example, the user has selected the power transformer numbered 123. The framework then queries the model and finds three classes having an associative relationship with the PowerTransformer class: Measurement, MaintenanceIntervention and PowerStationAndTransformerAssociation. Those three classes will be fetched but, for the sake of simplicity, this example only presents the Measurement class case. Its uniform resource identifier (URI) [33] and label have first been 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 property in the list of properties and its value for this individual, if any.

In the example in Figure 4, 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 to retrieve. 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.

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 individuals linked to the selected power transformer. Hence, if a new domain class is added, the application will automatically present it to the user.

- Class
  - URI: <hydroquebec:Measurement>
  - Label: "Measurement"
- List of properties
  - Prop 1
    - URI: <hydroquebec:Measurement#ETH
    - Label: "Ethanol_Concentration"
    - Range: <xsd:decimal>
    - Presentation information: "numberField"
  - Prop 2
    - URI: <hydroquebec:Measurement#METH
    - Label: "Methanol_Concentration"
    - Range: <xsd:decimal>
    - Presentation information: "numberField"
  - …
- List of individuals
  - Ind 1
    - Ethanol_Concentration: 127.6
    - Methanol_Concentration: 156.7
    - …
  - Ind 2
    - Ethanol_Concentration: 126.7
    - Methanol_Concentration: 157.6
    - …
- Access
  - Filter: <hydroquebec:PowerTransformer#123>
  - Path: <hydroquebec:Measurement#PowerTransformerURI>

Figure 4. Example of a JAVA generic object.

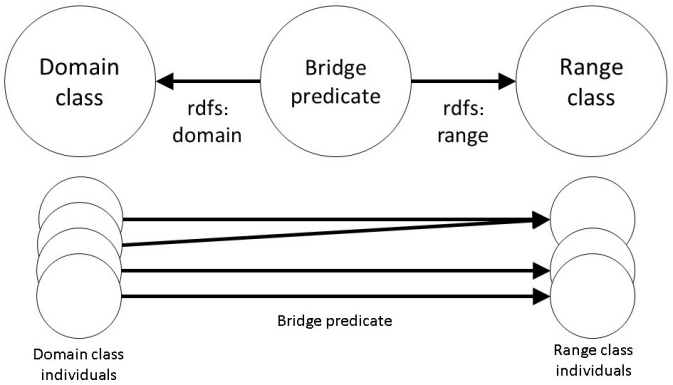


Figure 5. Graph representation of the range and domain classes in an associative relationship.

The application uses a tree to show the user a specific portion of the semantic graph (see Figure 7). In our case, the tree first shows all the power stations as folders that can be expanded to see the power transformers in each.

When the user selects a node (e.g., power transformer 123), the client UI sends a request to the AIS server using a generic process to dynamically gather the domain classes (e.g., the Measurement class) in relationship 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 [34] and sent to the UI.

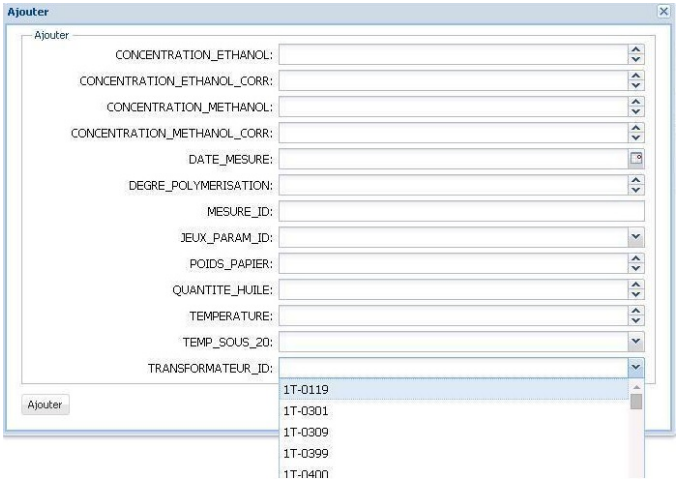


Figure 6. Example of individual CRUD form.

*b) System 2 - Generic triples system:* Using JAVA generic objects and transmitting them in the JSON format seems natural in an object-oriented paradigm. However, since the database is part of the semantic world, keeping the data in RDF format should also be appropriate.

An important difference between the first and the second system is that generic SELECT queries in the former are replaced by a generic CONSTRUCT query in the latter. The use of SELECT queries leads to the parsing of result sets in the object-oriented paradigm, i.e., the generic JAVA object. The CONSTRUCT query, on the other hand, leads to the creation of a new set of RDF statements. Using the Jena library, this set can be directly mapped into JSON-LD format.

By building the generic CONSTRUCT query, it was possible to extract the same data from the triplestore as when using System 1. This query was designed to produce an RDF graph with the same data structure as that of the JAVA generic object of System 1, so the client interface could use both transmission methods without any major change.

In order to transform the JSON-LD RDF graphs into a tree-like structure, the JSON-LD library for JavaScript was used [35]. This library enables the system to frame the RDF graph in a non-redundant data tree and to compact its URIs into keywords, thus presenting the information just as a normal JSON would do.

*3) After data transmission:* No matter which method is used, once transmission is complete and the data is sent to the client side, the UI will produce a 2D matrix for every class in the list (see Figure 7). These matrices show the information to the user using human readable labels. The user can then request CRUD operations on individuals represented in the matrices (see Figure 7).

Due to the genericity of the functions, changes made to the data model are immediately available to all AIS users. This genericity was obtained as discussed in the first and second principles of Table I. From then on, every request will retrieve individuals and classes from the new model with no need to recompile the client or server. This is because both the data model and its instances are queried. In addition to adapting to any model, a request used in runtime will inspect the actual version of the model.

In the current state of the framework implementation, if changes are made in the T-Box, either by modifying the 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 model. Without code refactoring, no other changes are possible in this implementation.

The main presentation tree does not grant access to every class in the semantic graph. Therefore, the UI has been given other access points, from which the user can request directly 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 class mechanism. Either the same generic JAVA object or generic triples model can be used, but they do not contain any access information. The same CRUD operations can still be performed on individuals.

*4) The CRUD services:* Our framework allows CRUD services only on individuals, not on classes or properties. Other means are used to edit the conceptual model. Further work will be done to allow modeling of the T-Box from the UI, either by adding capabilities to the OB or by developing ontology edition components usable in any AIS made with the AAB. 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 or the generic triples model.

In order to help the user and validate the input, a presentation knowledge base comprising the various 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 or the generic triples model. It indicates how to establish every entry field of the forms. Those forms are constructed dynamically, adapting the user's interaction options based on the values of properties according to the presentation knowledge base information. This dynamic type retrieval is in line with conceptual independence principle 4 in Table I.

In further developments, mechanisms will be designed to automatically link domain ontology properties to presentation ontology individuals. Some ontologies contain semantics, such as Enumeration or Bag, that can be used to predict the correct entry field's type for a given property. Enumeration, for instance, can be represented as a list of individuals that may be selected by the user. In general, the range of a property is a good indicator of the required type of a given entry field. Finally, functions will be implemented to allow the user to change the type of the entry field at runtime.

In the current state of the framework, four types of entry fields are implemented: numerical field, text field, list field and date field. Upon expansion, the list field requests 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 exist in the presentation knowledge base so the forms can specify to the user the required fields, if any.

*5) 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

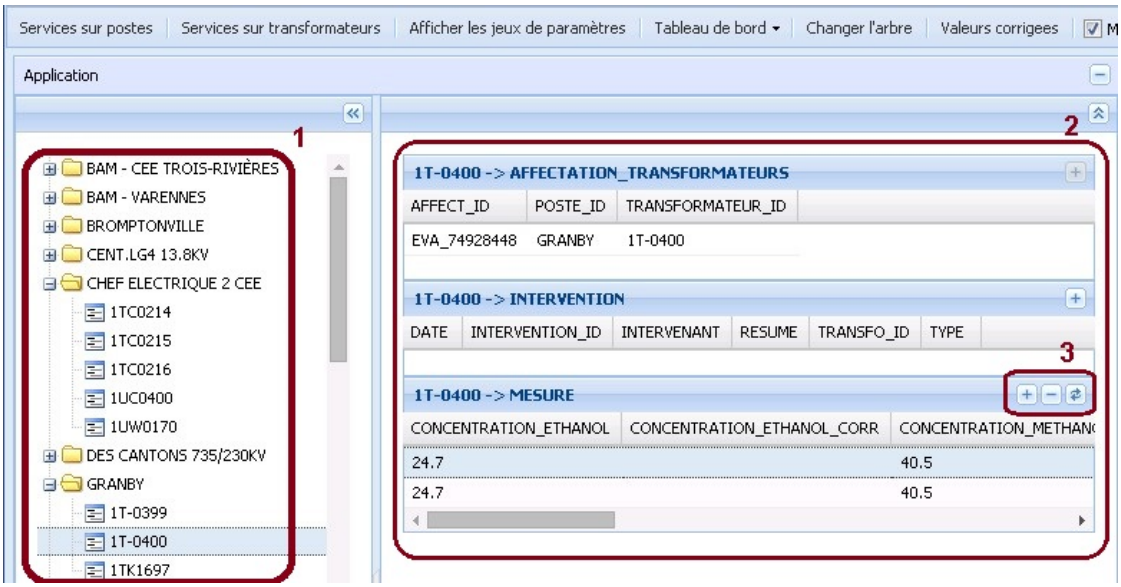


Figure 7. Application -(1) tree view, (2) matrices and (3) buttons for CRUD operations.

properties are used to specify the association between the graphic elements such as X-axis data, Y-axis data, labels, etc. The axes are linked to domain ontology properties.

### C. Organization of knowledge bases

This use-case application required the use of multiple stores to distinguish between domain information, presentation information, conceptual-model knowledge, individuals, editable and non-editable information. The application requires that all this information integrate seamlessly on the UI. When information is requested, a virtual aggregation is done, depending on which data sources the user wants to visualize.

The semantic graphs are organized as follows: the T-Box semantic models, one for the presentation knowledge and one for the domain knowledge, are separated from their respective A-Box semantic models (see Figure 8).

The presentation T-Box graph represents knowledge that will be common to all applications using the framework. Its A-Box contains form entry field types and application-specific graphics.

The domain T-Box graph represents knowledge specific to the application domain of expertise. Its A-Box is divided into four semantic graphs. It is first split into two graphs since data may come from two sources: the organization RDB or users themselves. For security reasons, it is impossible to apply changes to the organization RDB from the application. The A-Box is hence split into two graphs depending on the origin of the data: a non-editable one with data from the organization RDB and an editable one with user-created data. It is then easy to synchronize the non-editable graph to keep it updated, while not losing any user-created information. Moreover, this division enables the two data sets to be displayed independently, but only if another separation is created.

Independent data can be presented on their own to the user but other data depend upon these independent data to be shown. Therefore, a division relative to the dependence of the information is needed. Dependent A-Box individuals are

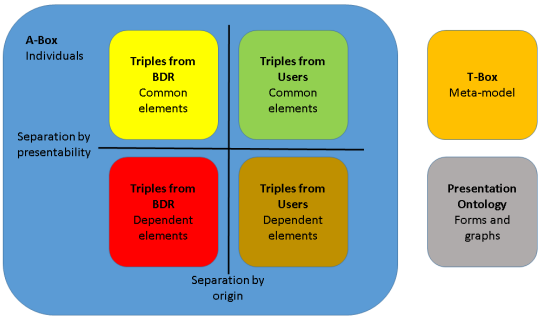


Figure 8. Organization of semantic graphs.

located in one graph and independent individuals are located in another. For example, if the user only wants to see the user-entered power transformer measurements imported from the organization database, information from both sources will be needed. Holding the power transformers in the independent graphs and the measurements in the dependent one will allow the display of user-entered measurements exclusively, even the ones describing power transformers held in the RDB. When CRUD services are used, a service is called upon to indicate on the form whether data can be modified or suppressed, depending on its source.

This organization of the A-Box in four semantic graphs was established for convenience, but it would be beneficial for this mechanism to be generic to the many applications used in an organization where information comes from various sources with different security constraints. Further testing must be done to establish a completely generic methodology.

## IV. Results

The OBAISF has been used to create a client-server decision support application. Using the generic services of the

framework, the classes and properties in the conceptual model can be modified directly in the triplestore without affecting the application. The UI presentation will adjust automatically according to the latest update of the conceptual model, since the request queries the semantic graph dynamically. The proposed framework supports all CRUD operations to be performed on individuals. Moreover, the framework will query the conceptual model in the semantic graph for each request. This differs from a standard application where the conceptual model is taken into consideration only at compile time. The resulting application is ready to be put into production. Once in production, since it will be able to automatically adapt to conceptual model changes, it should easily evolve as the framework is extended.

As stated, the OBAISF will enable the AAB. The AAB will help to create many information systems to be used throughout Hydro-Québec and its research institute. This should shorten the time needed to create applications. In research institute context where new knowledge in the domain of electric utilities is constantly emerging, auto-adaptive applications should require minimal time to maintain, thus reducing the time between discoveries and their practical implementation. As seen in Table I, RDF technologies have inherent properties facilitating the design of adaptive applications.

The adaptiveness of those applications is in large part due to the way the database is queried. Of the various ways that could have been used to obtain the data in a generic manner, the two below have been tested and compared.

### A. Results - Comparison of two communication systems

Two client-server communication systems were tested on the same virtual machine using 16 gigabytes of RAM and four cores. The first involves a generic JAVA object populated by generic SPARQL queries and automatically transformed to the JSON data format. The second uses another generic SPARQL query but to construct triple models sent to the client in the JSON-LD data format. Both communication systems manage to transmit the same data, formatted in the same structure, in a generic manner, thus bringing the same flexibility to the applications. Except for the communication systems, all the other methods of the AIS were shared between the two configurations. The two communication systems were compared for speed, code length and possibilities.

In our implementation, the generic JAVA object system retrieved small data sets faster, but became slower than the JSON-LD system as the data sets increased in size. This is primarily because the generic JAVA object is populated by several SPARQL SELECT queries; whereas the JSON-LD system is populated by only one CONSTRUCT query, less efficient for processing small data sets but gaining efficiency with larger ones.

To be completely fair in comparing the speed of the two systems, an effort should be made to minimize the number of queries in the first, and to optimize all the queries in both. Even doing so, the first technique will always require more queries than the second to obtain the same results. This is because SPARQL SELECT query outputs are result sets organized as 2D matrices. In order to have meaningful result sets, it is useful to split the queries; failing to do so creates numerous near identical lines in 2D matrices due to data replication. The number of queries needed for the first system is 1 + 2c, where c is the number of classes describing the selected individual retrieved. CONSTRUCT query output being a set of triples, the results are in the form of an oriented graph. In the second system, a single CONSTRUCT query can thus yield all the required results in a meaningful manner.

Code length was compared between the two systems. Though not a really meaningful metric, it is a first attempt at characterizing the two systems. Further work is needed to calculate better metrics to account for code maintainability, readability and complexity when comparing systems. For code length, the JSON-LD system is more advantageous, requiring three times fewer lines on the server side to yield the same result. On the client side, the code is almost exactly the same in both cases with the exception of the use of the JSON-LD.JS library [35] for the second system. This library is used to transform the triples in the received JSON-LD into JSON format by first rearranging them into a tree shape by framing them according to a pre-defined frame and then compacting the properties' URI into pre-set keywords. These two operations result in a JSON object usable by any JSON-friendly library.

A major advantage of the second system is only visible when picturing the AAB. With this system, SPARQL queries used for application customization can be stored as a set of triples. As the visual queries made from the SQB are stored in the form of triples, they can be easily linked to the domain ontology by simply joining the query repository with the ontology. This enables automatic validation of applications upon modification of the ontology. Thus, the changes in the ontology that will not be tackled by the auto-adaptive properties of the OBAISF will still be automatically spotted, marked and sent to a power user or an administrator. The same mechanism is conceivable with the first system but would be more difficult to implement.

Another advantage of the second system is potential use of inference capabilities on the client side of the applications. While it has not yet been tested, bringing triples from the triplestore to the client side would make it possible to use RDF-friendly algorithms, like inferencing, without any communication with the database server. This could be useful in some use cases, like for portable applications.

### B. Limitations

The main limitation of the entire framework is how it explores the model at each request. It can now only retrieve individuals from classes that are one associative relationship away from a desired individual. Further work is required to find ways to expand this exploration, a crucial factor for the framework to be effective in large-scale ontologies. The OB should help to develop this mechanism rapidly. Associating classes and properties with annotations seems like an easy way to solve this problem but more sophisticated solutions may yield better results.

Tests must still be run to determine performance differences between a dynamic application such as the one we developed and a static one, and to observe the scaling potential. The resulting application from the proposed framework is not expected to perform as well as a similar application developed from a more conventional framework but the difference in performance has yet to be established. It will then be possible to evaluate over the long run how much the lower costs incurred

during the AIS development and maintenance processes offset any reduced performance.

### C. Lessons learned

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

AISs built using the AAB will fall into the MDE paradigm, in the sense that a meta-model language is used to describe all AISs independently of their domain and that by designing their model using WAO components will suffice to generate their code. The AIS will be considered M0; WAO, M1; OWL, M2; and RDFS, M3. This architecture mainly uses RDF for the platform-independent model (PIM) and JavaScript for the platform-specific language (PSL). By encapsulating the PSL into the PIM, i.e., by inserting application components inside the classes of an ontology, the transformation model can become generic. This transformation model will work independently from the PSL. Indeed, as long as the engine reading the WAO instances and transforming them into web applications uses generic functions, it can build them in any web-friendly language. This should also make it possible to easily change PSL components, thus making application maintenance processes even more flexible. This technique should also increase code reuse across the entire application-building process since all applications built with it can share the same components and processes. Lastly, this should reduce refactoring efforts.

## V. Conclusion and future work

As hypothesized, an AIS based on a triplestore is easier to implement than an AIS using XML to dynamize functions on an RDB. Many artifices must be considered to build an AIS from a RDB, something not required with semantic technologies, as seen from Table I. The use of a library to map the RDB into a triplestore appears a judicious way to quickly and easily achieve the conceptual independence needed in an AIS.

With the use of an RDF representation to store the information, generic SPARQL queries that can search any semantic graph for both conceptual knowledge and individual information are easily devised. This makes the AIS able to adapt to changes in the conceptual model and to be used for different application domains. The framework could also be used with evolutionary prototyping application development as the future AAB. 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 [25].

A deeper analysis is needed to determine which of the two communication systems to choose. So far, the second system seems more promising, mainly due to greater speed in processing large data sets and its capability to automatically spot the impact a modification to the ontology will have on the applications. More exhaustive tests are underway and should improve results.

Based on this proof of concept, the proposed AAB will be designed to use this OBAISF in order to build numerous AISs. Our goal is to produce interpreted applications with their code and processes maintained inside the domain ontology. Any instantiation of the WAO will be a custom aggregation of different generic components using the generic functions of the OBAISF. The last part of the AAB is building an engine capable of reading a WAO instantiation and generating the application on the fly. A technique to encode treatment of the data inside the ontology should also be developed.

Using the OBAISF through the AAB to build new applications will further test the approach. In doing so, new functions will be developed eventually leading to more complete AISs. Ideally, the AISs should be able to take advantage of all RDF, RDFS, and OWL semantics. The current OBAISF uses only RDFS semantics; adding OWL capabilities will make the use of inference reasoners possible. This will be a major benefit to the AAB, the reasoner supporting validation during the creation of applications.

In the current release, only individuals can be edited by the user through forms. Editing features on the meta-model should be enabled by adding an ontology editor on top of the OB and by incorporating these capabilities into the framework. The OB should eventually also be enhanced to enable the insertion of treatments of data inside the domain model. This will help reduce refactoring time and improve code reuse throughout the company.

The framework and the application demonstrate that an AIS can work easily and efficiently by capitalizing on RDF technology and its inherent properties. The future AAB should leverage the framework by giving power users an easy means to implement it in a wide variety of adaptive applications. Such systems can be useful in fast-evolving knowledge domains. They are fully in line with the AGILE development philosophy, allowing the data model to evolve freely at each iteration. Those considerations suggest that OBAISF 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, Arnaud Zinflou and Billel Redouane.

## References

[1] 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, Nice, France, July 2015, pp. 110–115.

[2] 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.

[3] J. Sequeda. Introduction to: Triplestores. [retrieved: 02, 2016]. [Online]. Available: http://www.dataversity.net/introduction-to-triplestores/ (2013)

[4] 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.

[5] 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.

[6] 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.

[7] T. Gherbi, D. Meslati, and I. Borne, "MDE between promises and challenges." in UKSim, D. Al-Dabass, Ed. IEEE Computer Society, 2009, pp. 152–155. [Online]. Available: http://dblp.uni-trier.de/db/conf/uksim/uksim2009.html#GherbiMB09

[8] D. C. Schmidt, "Model-driven engineering," IEEE Computer, vol. 39, no. 2, February 2006. [Online]. Available: http://www.truststc.org/pubs/30.html

[9] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing?" International Journal of Human-Computer Studies, vol. 43, no. 5–6, 1995, pp. 907 – 928. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1071581985710816

[10] O. M. Group. Executive overview model driven architecture. [retrieved: 02, 2016]. [Online]. Available: http://www.omg.org/mda/executive_overview.html/ (2014)

[11] W3C. Ontology driven architectures and potential uses of the semantic web in systems and software engineering. [retrieved: 02, 2016]. [Online]. Available: https://www.w3.org/2001/sw/BestPractices/SE/ODA/ (2006)

[12] J. Z. Pan, S. Staab, U. Aßmann, J. Ebert, and Y. Zhao, Eds., Ontology-Driven Software Development. Berlin: Springer, 2013.

[13] M. Zviedris, A. Romane, G. Barzdins, and K. Cerans, Semantic Technology: Third Joint International Conference, JIST 2013, Seoul, South Korea, November 28–30, 2013, Revised Selected Papers. Cham: Springer International Publishing, 2014, ch. Ontology-Based Information System, pp. 33–47. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-06826-8_3/

[14] J. Sowa, "Serious semantics, serious ontologies, panel," in Presented at Semantic Technology Conference (SEMTEC 2011), San Francisco, 2011.

[15] P. Horn, "Autonomic Computing: IBM's Perspective on the State of Information Technology," Tech. Rep., 2001.

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

[17] I. T. C. on Software Engineering. IEEE EASe 2014. [retrieved: 06, 2015]. [Online]. Available: http://tab.computer.org/aas/ease/2014/index.html (2014)

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

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

[20] 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.

[21] A. Berns and S. Ghosh, "Dissecting self-* properties," 2013 IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems, 2009, pp. 10–19.

[22] 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.

[23] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," ACM Trans. Auton. Adapt. Syst., vol. 4, no. 2, May 2009, pp. 14:1–14:42. [Online]. Available: http://doi.acm.org/10.1145/1516533.1516538

[24] P. A. Akiki, A. K. Bandara, and Y. Yu, "Adaptive model-driven user interface development systems," ACM Comput. Surv., vol. 47, no. 1, May 2014, pp. 9:1–9:33. [Online]. Available: http://doi.acm.org/10.1145/2597999

[25] 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.

[26] 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.

[27] D. Beckett. What does SPARQL stand for? [retrieved: 02, 2016]. [Online]. Available: https://lists.w3.org/Archives/Public/semantic-web/2011Oct/0041.html (2011)

[28] T. A. S. Foundation. Apache jena. [retrieved: 02, 2016]. [Online]. Available: https://jena.apache.org/ (2015)

[29] Sencha. Ext js 4.2. [retrieved: 02, 2016]. [Online]. Available: http://docs.sencha.com/extjs/4.2.2/ (2014)

[30] C. Bizer. D2RQ. [retrieved: 02, 2016]. [Online]. Available: http://http://d2rq.org/

[31] T. C. L. C. Inc. Definition of: RDF. [retrieved: 02, 2016]. [Online]. Available: http://www.pcmag.com/encyclopedia/term/50223/rdf (2016)

[32] W3C. Json-ld 1.0. [retrieved: 02, 2016]. [Online]. Available: https://www.w3.org/TR/json-ld/ (2014)

[33] T. Berners-Lee. Naming and addressing: Uris, urls, ... [retrieved: 02, 2016]. [Online]. Available: https://www.w3.org/Addressing/ (1993)

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

[35] W. J.-L. C. Group. JSON for linking data. [retrieved: 02, 2016]. [Online]. Available: http://json-ld.org/ (2014)