International Journal of Software Engineering
and Knowledge Engineering
Vol. 26, No. 1 (2016) 115–145

DOI: 10.1142/S0218194016500066

OAM: An Ontology Application Management Framework for Simplifying Ontology-Based Semantic Web Application Development

Marut Buranarach*, Thepchai Supnithi†, Ye Myat Thein‡
and Taneth Ruangrajitpakorn§

National Electronics and Computer Technology Center (NECTEC)
Pathumthani, Thailand
**marut.bur@nectec.or.th*
†*thepchai.sup@nectec.or.th*
‡*yemyatthein@gmail.com*
§*taneth.rua@nectec.or.th*

Thanyalak Rattanasawad

Department of Computer Engineering
Chulalongkorn University, Bangkok, Thailand
thanyalak.rattanasawad@gmail.com

Konlakorn Wongpatikaseree¶, Azman Osman Lim‖ and
Yasuo Tan**

Japan Advanced Institute of Science and Technology (JAIST)
Ishikawa Prefecture, Japan
¶*w.konlakorn@jaist.ac.jp*
‖*aolim@jaist.ac.jp*
***ytan@jaist.ac.jp*

Anunchai Assawamakin

Department of Pharmacology
Mahidol University, Bangkok, Thailand
anunchai.asa@mahidol.ac.th

Received 27 May 2014
Revised 23 December 2014
Accepted 8 March 2015

Although the Semantic Web data standards are established, ontology-based applications built on the standards are relatively limited. This is partly due to high learning curve and efforts demanded in building ontology-based Semantic Web applications. In this paper, we describe an ontology application management (OAM) framework that aims to simplify creation and adoption of ontology-based application that is based on the Semantic Web technology. OAM

introduces an intermediate layer between user application and programming and development environment in order to support ontology-based data publishing and access, abstraction and interoperability. The framework focuses on providing reusable and configurable data and application templates, which allow the users to create the applications without programming skill required. Three forms of templates are introduced: database to ontology mapping configuration, recommendation rule and application templates. We describe two case studies that adopted the framework: activity recognition in smart home domain and thalassemia clinical support system, and how the framework was used in simplifying development in both projects. In addition, we provide some performance evaluation results to show that, by limiting expressiveness of the rule language, a specialized form of recommendation processor can be developed for more efficient performance. Some advantages and limitations of the application framework in ontology-based applications are also discussed.

Keywords: Semantic Web application framework; ontology application framework; ontology-based data publishing and access; knowledge-based application development tools.

1. Introduction

With the Semantic Web data standards being established, some organizations and initiatives started creating and sharing their data in the Resource Description Framework (RDF) format, aka. Linked data initiative [1]. Further, domain knowledge in different areas has been increasingly captured in ontology form that can be shared as the Web Ontology Language (OWL) data that can be linked with the RDF data. Although creation of the Semantic Web data rapidly grows, e.g. the Linked data cloud [1], ontology-based applications built on the Semantic Web data are relatively limited. This is partly due to high learning curve and efforts demanded in building ontology-based Semantic Web applications. To facilitate development of such applications, development tools should allow application developers to focus more on domain problems and knowledge rather than implementation details. Put another way, application development tools should not only be designed for technologists but also for researchers or domain experts who are non-technology experts. Simplifying development of Semantic Web applications is important in promoting Linked Data publishing and Semantic Web application deployment [2].

Much of software development effort is caused by re-invention of core concepts and components in building applications from scratch [3]. An application framework is a reusable, 'semi-complete' application that can be specialized to produce custom applications [3]. Unlike software library, an application framework is usually more targeted for a specific class of software [4]. This paper describes an application framework designed for ontology application that introduces an intermediate layer between user application and programming and development environment. It focuses on supporting ontology-based data publishing and access, abstraction and interoperability. The framework provides reusable and configurable data and application templates, which allow the users to create ontology applications without programming skills required.

The Ontology Application Management (OAM) framework is a development platform for simplifying creation and adoption of ontology-based Semantic Web application, focusing on semantic search and recommender system applications.

OAM introduces specialized templates for database to ontology data mapping, recommendation rule as well as application templates. Our framework is different from existing Semantic Web application frameworks in that it does not require user's programming skill in building a Semantic Web application prototype. Thus, it is suitable for researchers who want to experiment on research ideas that can be realized by means of the Semantic Web technology. Application template is typically ideal for rapid prototyping and hypotheses testing. The framework also provides application-level APIs to support a more advanced application development.

In this paper, we describe the design and implementation of the OAM framework focusing on three main components: database to ontology mapping, recommendation management and application templates and APIs. The community-driven software development model used in the tool development is also elaborated. Two research projects that highlight the utility of the framework are discussed: human activity recognition in smart home environments and an ontology-based clinical support system. The paper is organized as follows. Section 2 discusses related work and systems. Section 3 focuses on some unique design approaches and implementation details of the OAM framework. Section 4 discusses two case studies that adopted the OAM framework. In Sec. 5, the potential roles and benefits of the framework are discussed.

2. Background and Related Work

2.1. *Ontology-based data publishing and access*

Data integration is important for Semantic Web application development as the data can come from different sources. There are three approaches in using ontology to augment data integration: single, multiple and hybrid ontology approach [5]. In the single ontology approach, all information sources are related to one global ontology, which is used as a global schema that provides a virtual view of the underlying sources which store the real data [5]. An example is the Ontology-based Data Access (OBDA) approach [6] which uses ontology as the global unified view for accessing the data stored at the sources. Mappings are required to specify the semantic correspondence between the unified view of the domain and the data stored at the sources. In the multiple ontology approach, each information source is described by its own ontology or "local ontology". The Linked Data initiative is an example of the multiple ontology approach. The Linked Data sources either use their own schemata or use a mixture of terms from existing, well-known vocabularies together with self-defined terms specific to the particular data source [1]. Thus, schema mapping and data fusion are often needed for mapping of terms from different vocabularies to the target schema [7], aka the hybrid ontology approach. The Linked Data Integration Framework (LDIF) [7] is an example of the hybrid ontology approach applied to integrating the Linked Data sources. LDIF provides a mapping language to allow for translating data from the various vocabularies into a consistent local target vocabulary.

A data integration system typically aims to give users or applications the ability to query information from different sources and to return the results in a uniform way. Architecture of such a system can be categorized into two types: Virtual Integrated Systems and Materialized systems [8]. OBDA systems are typically in the former category including MASTRO [9], OPTIQUE [10], etc. The queries formulated using ontology terms are automatically translated using mappings into the queries over the data sources. The main advantage of this approach is that the data source is autonomous and always up to date. In the materialized systems approach, redundant copy of the source data is extracted from the source data periodically. The global schema must be defined to allow integration of the data from different sources. Data warehousing and Linked Data integration exemplify such an integration approach. Although this approach benefits from simple integration and query method, which does not require query translation, the maintenance of the up-to-datedness of the replicated data is more difficult [8]. Applications of this approach are typically analysis-driven applications rather than transaction-driven applications.

The OAM framework adopts the single ontology and the materialized system approaches. A global schema or domain ontology is required in building an OAM application. The OAM application operates on the replicated copy of the data in the RDF format published from the original data sources. Unlike OBDA systems, ontology-based data mappings are only used for data publishing but not when querying. This has an advantage of simpler querying, which is based only on SPARQL. The framework is more suitable for supporting analysis-driven applications where maintenance of up-to-datedness of the replicated RDF data is less complex.

2.2. *Semantic Web application framework*

Much of software development effort is caused by re-invention of core concepts and components in building applications from scratch [3]. An application framework is a reusable, 'semi-complete' application that can be specialized to produce custom applications [3]. Unlike software library or toolkit which is a set of related and reusable classes designed to provide useful, general-purpose functionality, a framework is a set of cooperating classes that make up a reusable design for a specific class of software [4]. Thus, by adopting a framework, an application can be customized by creating application specific subclasses of abstract classes of the framework.

Application framework for the Semantic Web domain may be grouped into two categories: programming environment and development environment. Although distinctions between both categories are sometimes blurred, less programming effort is normally required for the latter category, which supports high-level abstraction. Development environment tools typically aim at reducing the user effort by providing facilities supporting common tasks required in developing Semantic Web

applications, e.g. Protégé,[a] AllegroGraph,[b] Jena framework [11], TopBraid Suite,[c] etc. Classification and definitions of components for Semantic Web application framework are provided in [12]. There are some application frameworks that aim to simplify and promote rapid development of Semantic Web applications including the Semantic Web Application Framework (SWAF) [13], Callimachus,[d] etc.

A specific kind of Semantic Web application framework focuses on supporting building Web applications on top of Linked Data or "Linked Data-driven applications" [14]. Development of Linked data applications can utilize similar tools to development of Semantic Web applications. Some specific issues to Linked Data applications include data publishing, which turns any kind of structured data into RDF and interlinks the dataset, and consumption process, which consists of discovery, access and processing [15]. Functions of tools and systems designed to simplify Linked Data-driven application development process range from supporting Linked Data publishing from existing data sources and consume the data over Linked Data sources, such as LMF [16], and supporting Linked Data integration, mash-up, and analysis framework, such as LDIF [7], Sigma[e] and Callimachus.

The OAM framework is different from existing Semantic Web application frameworks in its proposed uses of reusable and configurable data and application templates in the intermediate layer between user application and programming and development environment. The framework provides some graphical user interfaces to allow the user to manipulate the data and application templates, in customizing the application. Thus, it does not require user's programming skill in building an ontology-based application prototype. The framework differs from Linked Data-driven application platforms in that it requires domain ontology in publishing and consuming the RDF data. Thus, it only supports publishing and consuming the RDF data that come from the sources that agree on the same domain ontology. Some applications include enterprise or domain-specific applications that require common domain ontology in publishing, integrating and consuming the data from different data sources.

3. Design and Implementation of the OAM Framework

3.1. *Community-driven software tool development*

A community-driven approach was applied to the development of the OAM software tool[f] [17]. The approach focused on conducting related activities to support user/developer community in augmenting software tool development. The activities included both adoption and development support activities. In this approach, user

[a] http://protege.stanford.edu/
[b] http://www.franz.com/agraph/allegrograph/
[c] http://www.topquadrant.com/products/
[d] http://callimachusproject.org/
[e] http://sig.ma/
[f] http://lst.nectec.or.th/oam/

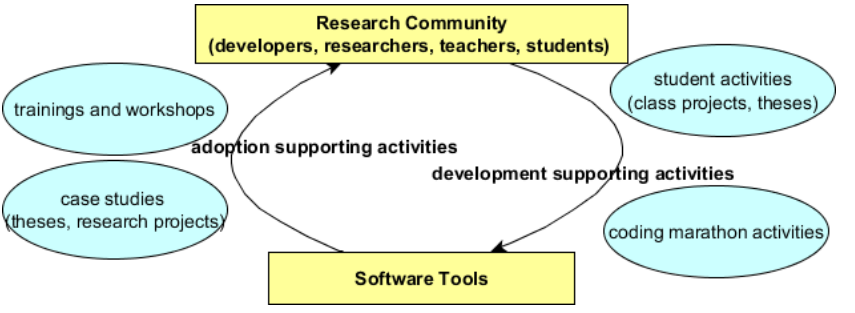


Fig. 1. A community-driven software tool development cycle.

involvement and support is more clearly targeted. Our experience has shown that this approach helped to promote communication and co-creation between the community and researchers, which subsequently facilitated software tool evolvement. The approach can be illustrated as a software development cycle as shown in Fig. 1.

In this model, contributions from user/ developer community are significant to the development and adoption of a software tool. In supporting tool adoption, training and workshop activities can be organized to introduce the tool to the community. In addition, some case studies should be provided to demonstrate application potentials of the tool. Related student theses or research projects that adopt the tool should be set up. The community users who adopted the tool can help to provide feedbacks, testing and evaluation results, which can contribute to gathering additional requirements for improving the tool. In supporting tool development, university teachers can integrate some parts of the tool development as student assignments for class projects or student theses. In addition, coding marathon activity, which is popularly adopted in opensource software development, can be organized to promote collaborative efforts between researchers and developers in improving design and implementation of the tool.

Since the initial release of the OAM software tool in 2010, we have conducted user training sessions to introduce the tool to the Semantic Web interest group in Thailand. Each session comprised of two hand-on trainings: ontology development[g] and ontology application development. Feedbacks from the participated users, who were not only technologists but also domain experts who do not have programming skill, were gathered as user requirements to guide the tool improvement. Since the number of community users is small, handling the user requirements can be more focused with continuous follow-up. In addition, some university teachers, who have participated in the trainings, have taken the requirements and assigned them as student class projects. In addition, we organized a coding marathon activity which involved students, researchers and professional developers. These supporting

[g]Hozo ontology editor (http://www.hozo.jp/) was used in the ontology development trainings.

activities have collectively contributed to improving the tool both in terms of functionalities and user interface design. Further, several projects and theses were set up and adopted the tool, which helped to demonstrate potential applications of the tool.

3.2. *Conceptual architecture*

The Ontology Application Management (OAM) framework is an application development platform aims to simplify creation and adoption of an ontology-based Semantic Web application. The application framework differs from the existing tools in two main aspects. First, it is an integrated platform that supports both RDF data publishing from databases based on domain ontology and processing of the published data in ontology-based Semantic Web applications, i.e. semantic search and recommender system applications. Second, the framework provides some reusable and configurable data and application templates that can be customized for different domain ontologies using configuration GUIs. Thus, it does not require user's programming skill in building an ontology-based Semantic Web application prototype.

The OAM framework introduces intermediate layers between user application and existing Semantic Web programming and development environment. Design of the framework is based on three principles: ontology-based data publishing and access, abstraction and interoperability. OAM requires ontology as a central structure for publishing RDF data from database and as a means to access the published RDF data. The layers introduced by OAM aim to hide complexity of the underlying Semantic Web data standards and models. The framework is designed to be independent of the underlying implemented systems. Thus, wrapper architecture is required for interoperation with different data formats and systems. Figure 2 shows a layered architecture of the OAM framework.

Currently, the framework is implemented on top of existing Semantic Web data and application platforms, i.e. D2RQ [18], Jena's RDF data storage and Jena's reasoning engine [19]. Additional wrappers for different systems may be added in the future. The OAM framework adds relational database to ontology data mapping, recommendation and application templates on top of these systems. The user can use the provided Web-based management tools in creating and managing an ontology-based Semantic Web application. OAM also provides Java API to support a more advanced application development.

3.3. *Database to ontology data mapping framework*

Ontology-based application development often involves manipulation of instance data. There are typically two methods in creating instances for ontology classes. The first method is to manually construct an instance and define its attribute values based on a class. This is typically done using instance editor provided in ontology

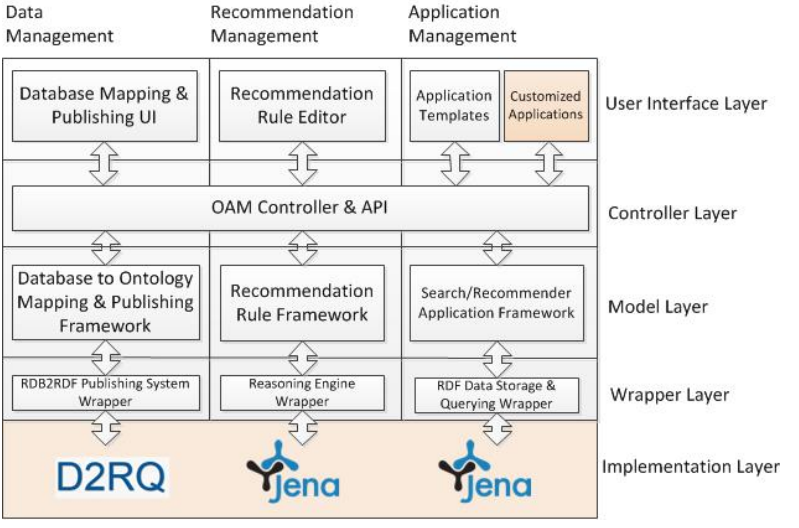


Fig. 2. Layered architecture of the OAM framework.

development tool. The second method is to create instances from some existing information sources, such as database records. This normally requires mapping process between the existing database schema and ontology structure [20]. After the mapping process, a database record can be transformed into a class instance. This method is most suitable when an organization already stored the data in some databases. Our framework focuses only on the latter approach in creating instance data from existing databases.

OAM's data mapping tool supports schema mapping between OWL ontology and a relational database source. The schema mapping configuration template in OAM is a specialized form of the D2RQ Mapping language [21]. OAM implements class-based mapping configuration, i.e. each chosen class in ontology must have its own configuration. The mapping configuration for each class involves two parts: class-table mapping and property-column mapping.

3.3.1. *Class-table mapping*

In class-table mapping, the user can define mapping between an ontology class and a database table. A property of the ontology class must be chosen to map with the primary key column of the table. This configuration allows the unique identifier of each database record to be transformed to local name identifier of each instance of the class.

In addition, the user can optionally define subclass mapping between subclasses of the class and some records of the database table. Specifically, the user can specify each attribute value of a table column as a determiner for instance of each subclass.

Without subclass mapping, all database records of the table will be transformed to instances of the class.

3.3.2. *Property-column mapping*

In property-column mapping, two types of mapping can be defined: data property and object property mappings. The user can define mapping between each property of a class, i.e. either data property or object property, with a column of a table with an optional join condition. The join condition is required when the column is in a different table from that defined in class-table mapping. The property mapping configuration supports one-to-one, one-to-many, many-to-one and many-to-many relationship types in databases.

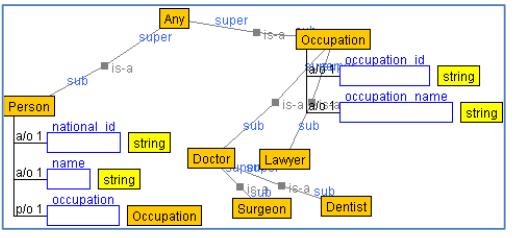
One-to-one relationship mapping: In this relationship type, a property of the class is mapped with a column of the same table defined in the class-table mapping. In this case, no SQL join statement is needed for the mapping.

One-to-many, Many-to-one and Many-to-many relationship mapping: In these relationship types, a property of the class is mapped with the column of a different table from that defined in class-table mapping. In this case, the user must define an SQL join condition that may involve some intermediate tables for the mapping.

3.3.3. *Example of database schema to ontology mapping*

Figure 3 shows an example of class-based mapping configuration used in the OAM Framework. An example ontology created using Hozo ontology editor, as shown in Fig. 3(a), consists of two main classes: Person and Occupation. The example database consists of three tables: persons, positions and professions (Fig. 3(b)). Two mapping configurations are required — each mapping configuration for each class. The class-table mappings for Person and Occupation classes with the persons and professions tables are straightforward because the classes and their unique properties can be mapped with the tables and their primary keys respectively (Fig. 3(c)). Finally, subclass mapping is required for the Occupation class. For example, the records of the professions table, whose profession names are equal to "attorney" or "solicitor", are assigned as instances of the Lawyer class (Fig. 3(d)). The Person class contains no subclass thus all the records of the person table are automatically assigned as instances of the Person class.

The data property-column mappings for both classes and their respective tables are also straightforward. The person name and occupation name can be mapped to the columns of their respective tables, i.e. one-to-one relationship type. Thus, no table joining is required (Fig. 3(e)). The object property-column mapping for the occupation property of Person class required join conditions over the three tables: persons, positions and professions, i.e. many-to-one relationship type (Fig. 3(f)). In current implementation, the OAM mapping configuration is subsequently converted



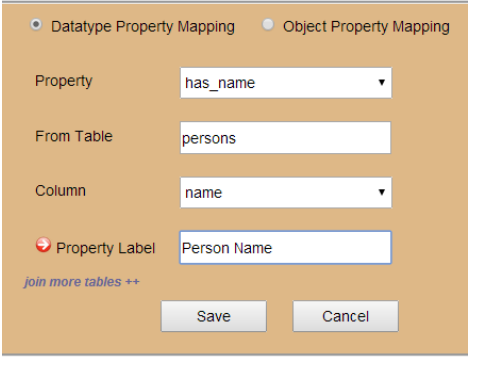
(a) Ontology

persons		
pid	name	pos_id
1	A	1
...	...	...

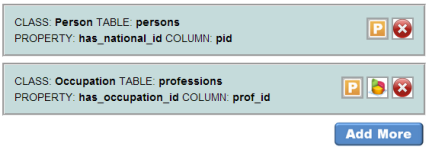
professions	
prof_id	name
1	neurosurgeon
2	plastic surgeon
3	orthodentist
4	attorney
5	solicitor

positions			
position_id	name	salary	profession_id
1	X	10,000	2
...	...	...	...

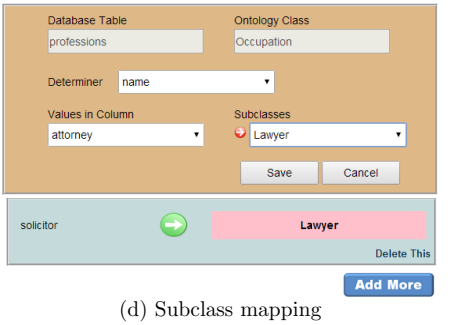
(b) Database



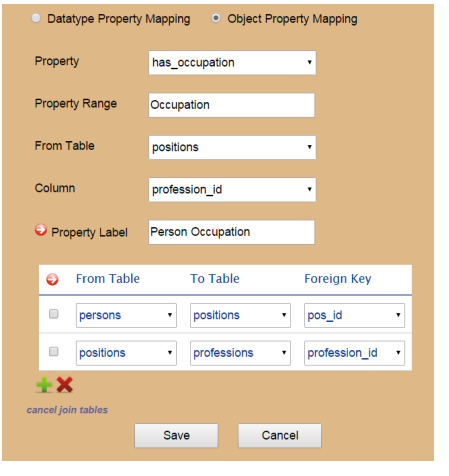
(e) Data property to column mapping for Person class



(c) Class-table mapping



(d) Subclass mapping for Occupation class



(f) Object property to column mapping for Person class

Fig. 3. An example of OAM class-based mapping configuration.

to the D2RQ mapping language which is used by the D2RQ system to generate the resulted instance data in RDF format.

3.4. *Recommendation management framework*

One of the applications of the OAM framework is supporting recommender system application [22] development. Recommender system is a type of system that

generates meaningful recommendations to support user's decision. Development of recommender system for the Semantic Web data typically requires ontology, rules and rule-based inference engine to be applied over the RDF data. To facilitate development of recommender applications, the OAM framework introduces a recommendation template which is a specific form of rule language that provides high-level abstraction in generating recommendation results. Recommendation rules can be created based on the template using recommendation editor that hides complexity of rule language syntax.

By adopting the recommendation template approach, implementations of rule editor and inference engine are less complex comparing to those of the generic rule editor and rule-based inference engine. In the case study section, we will compare the performance of the two implementation approaches for generating recommendation results: generic rule-based inference engine and our SPARQL-based implementation of inference engine designed for processing rules provided in the template. The results suggested that, by limiting expressiveness of recommendation rules to the template, a specialized form of recommendation processor can be developed for more efficient system performance.

3.4.1. *Recommendation template*

The OAM framework focuses on simplifying creation and management of recommendation rules based on a recommendation template. The recommendation template consists of three main components: "recommendation", "recommend-of" and "recommend-to" resources, as shown in Fig. 4. A recommendation rule is created in a two-step process: create recommendation and link recommendation. Creating recommendation will create an instance of a recommendation container class, e.g. "Promotion" where the user can define conditions of class instances to be the recommendations, e.g. "Product". Linking recommendation allows the user to define

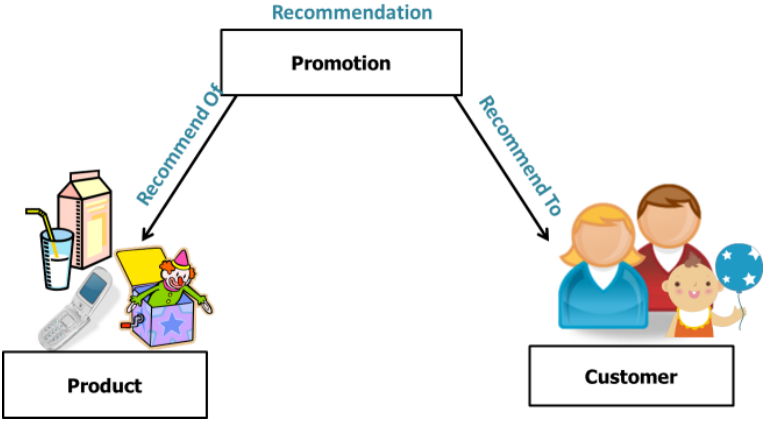


Fig. 4. Components of the OAM recommendation template.

conditions of class instances to recommendation receivers, e.g. "Customer". The framework facilitates the user to create such business logics using a form-based user interface that hides complexity of the rule syntax to be processed by reasoning engine.

The structure of a recommendation rule created based on the recommendation template can be summarized as follows.

- **Recommendation rule:** A recommendation rule consists of rule name, two condition sets for matching each part of recommendation rules: recommendations and recommendation receivers, and a property of the receiver class for receiving the recommendations.
- **Condition set:** A condition set consists of condition set name, matching class, a class of the individuals to be matched, and conditions.
- **Condition:** A condition for matching individuals. It consists of a property chain, an operator, and a value object.
- **Property Chain:** A property chain is an ordered series of one or more object-type properties. Each property links to an individual of the object of a triple recursively like joining many tables in relational database.
- **Operator:** An operator for matching or comparing between the condition's value object and a triple's object. The operators supported are mathematics comparison operator (=, >, <, >=, <=), string operator (=, contains). They can be used in the forms: <property> <operator> <value> and <property> <operator> <property>. RDF comparison operator (type) can be used in the form <property> <operator> <class>.
- **Object:** Object of a condition. It is in three types: literal value, URI, and property chain node value. A literal value is an RDF literal with a specified data-type. A property chain node is a node that refers to value of another property chain, allowing it to be operated with the literal value of the current condition.

Recommendation rules are stored in an intermediate format using JSON and can also be exported to a number of interchange formats.

To exemplify the format of the rule template, we make an example of a recommendation "Recommend the products which have discount rate more than 10 percent, and have price more than 10 dollars but less than 15 dollars, to the customers which have bought products from the store more than 20 times".

Figure 5 illustrates elements of the recommendation rule and linked condition sets in JSON format. Condition sets are illustrated in Fig. 6. The user can use the

```
"ClearancePromotion": {
  recOfRule: "DiscountedProduct",
  recToRule: "FrequentCustomer",
  recProperty: "suggestedProducts" }
```

Fig. 5. Elements of recommendation rule and linked condition sets.

```
"DiscountedProduct": {
matchingClass: "http://ex.org/Product", conditions: [
{ propertyChain: ["http://ex.org/discountRate"], operator: ">", object: ["float","10.00"]},
{ propertyChain: ["http://ex.org/price"], operator: ">", object: ["float", "10.00"]},
{ propertyChain: ["http://ex.org/price"], operator: "<", object: ["float", "15.00"]]}

"FrequentCustomer": {
matchingClass: "http://ex.org/Customer",
conditions: [{ propertyChain: ["http://ex.org/boughtRecord", "http://ex.org/boughtTimes"],
operator: ">", object: ["int", "20"]]}
```

Fig. 6. Example of matching condition sets.

provided rule editor to create recommendation rules based on the template that hides the complexity of the created rule syntax. The internal recommendation rule format can be serialized to a number of rule language formats for interchange with different rule-based inference engines.

3.4.2. *System architecture*

The recommendation management framework is designed based on three main principles: abstraction, extensibility, and interoperability. Abstraction is achieved by providing higher level of constructs than those provided by the RDF data model for building Semantic Web-based recommender applications. Extensibility is achieved by designing each module that is independent of the underlying implemented systems, i.e. inference engines and databases. Interoperability is achieved by allowing exports to the standard rule formats to enable interchanging and integration with other rule-based systems.

Figure 7 illustrates the system architecture of the recommendation management module. The follows describe details of the related system components.

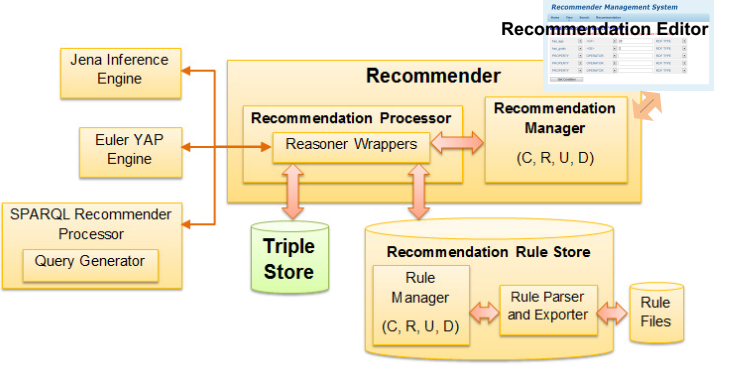


Fig. 7. Architecture of the OAM recommendation management module.

- **Triple Store:** The module provides database-independent interfaces of common functions for querying RDF database. This module allows the user to perform arbitrary queries and also provides query generator with some query templates.
- **Recommender:** The module for recommendation processing which contains sub-modules for managing and processing recommendation data: recommendation rules and recommendation instances.
- **Recommendation Rule Manager:** The module for recommendation rule management, which enables the user to create, view, edit, and delete recommendation rules, import other rules in format of the rule template, and export rules to interchange formats, e.g. RDF, JSON, and RIF [23], and a format of supported reasoner, such as Jena rule [19] and Notation3 [24].
- **Recommendation Processor:** The sub-module which provides common interfaces and encapsulates algorithms of recommendation creation, and recommendation results generation.

3.4.3. *Implementation of the recommendation processor*

Recommendation results are generated and linked with the related resources by passing recommendation rules and data to the recommendation processor. In our framework, the implementations are categorized into two approaches: rule-based inference engine approach, and SPARQL-based approach, which are described as follows.

- **Rule-based inference engine approach:** We created implementations using two different rule-based reasoners: Jena inference engine [19] and Euler YAP Engine [25]. The recommendation processor generated recommendation rules in form of if-then rule in the supported format of each reasoner. Recommendation results are created by means of rule-based reasoning. Figure 8 shows an example of the generated recommendation rule in Jena rule syntax.
- **SPARQL-based approach:** This approach adopts a SPARQL engine as the recommendation processor [26][27], and generates recommendation rules in the

```
@prefix : <http://ex.org/>. @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
(?recOf rdf:type :Product) (?recOf :discountRate ?v1) greaterThan(?v1, 10.00)
(?recOf :price ?v2) greaterThan(?v2, 10.00) lessThan(?v2, 15.00) -> (:ProductRec_1 rdf:type
:Product_Rec) (:ProductRec_1 :hasInstances ?recOf) (:ProductRec_1 :hasRecName 'ProductSetA')
(:ProductRec_1 :hasRecId '1')

(?recTo rdf:type :Customer) (?recOf :boughtRecord ?v3)
(?v3 :boughtTimes ?v4) greaterThan(?v4, 20) -> (?recTo :hasSuggestedProducts :ProductRec_1)
```

Fig. 8. Jena rule syntax of the recommendation rule.

```
PREFIX : <http://ex.org/>. PREFIX rdf: < http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
INSERT {
  ?recTo :hasSuggestedProducts :ProductRec_1.
  :ProductRec_1 rdf:type :Product_Rec.
  :ProductRec_1 :hasInstances ?recOf.
  :ProductRec_1 :hasRecName "ProductSetA".
  :ProductRec_1 :hasRecId "1". }
WHERE {
  ?recOf rdf:type :Product.      ?recOf :hasDiscountRate ?v1.      ?recOf :price ?v2.
  ?recTo rdf:type :Customer.    ?recTo :boughtRecord ?v3.          ?v3 :boughtTimes ?v4.
  FILTER(?v1 > 10.00)  FILTER(?v2 > 10.00)  FILTER(?v2 < 15.00)  FILTER(?v4 > 20)}
```

Fig. 9. SPARQL update syntax of the recommendation rule.

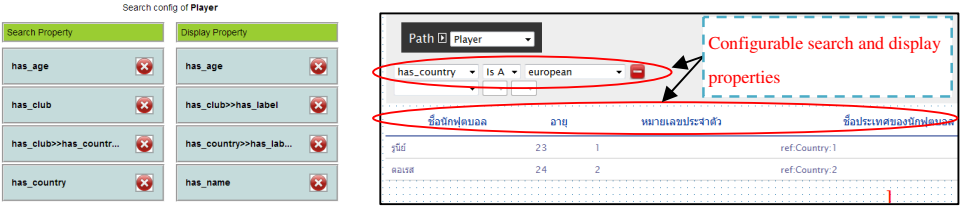
form of SPARQL queries and updates. Figure 9 shows example syntax of an SPARQL update for generating recommendation results. Currently, it can only support rules that generate results which do not fire another rule.

3.5. *Application template and programming interface*

3.5.1. *Application template*

OAM allows the user to apply a provided application template to the RDF data published based on an ontology to create an ontology-based application. Using application template, the user only needs to define application configuration and does not need programming skill. This is suitable for researchers who want to experiment on research ideas that can be realized by means of the Semantic Web technology. Application template is typically ideal for rapid prototyping and hypotheses testing.

Figure 10(b) shows a semantic search application template, which provides a faceted search [28] interface. Using the provided interface, the end-user can select a class in the ontology to search for its instance data and define some search property conditions. A search allows value comparison using both string and number



(a) Search application configuration (b) Search application template

Fig. 10. Example of a faceted search application template and configuration.

comparators and semantic-based comparators, i.e. IS-A (or type). The user's faceted search condition is automatically transformed to a SPARQL query for retrieving the instance data from the triple store. Figure 10(a) shows a search application configuration which allows the user to customize properties in search conditions and displayed search results. A property can be defined as a reference property, which is a property of another class that is referenced by means of a referring property. The reference property is denoted using the '≫' symbol.

3.5.2. *Application programming interface*

Application templates and custom applications are built using the OAM API. The API provides high-level abstraction that hides complexity of the Semantic Web data standards and models. Class diagram of the API is shown in Fig. 11.

Seven main classes and their main functions are shown: *ApplicationTemplate*, *Recommender*, *SemanticSearch*, *Mapping*, *RuleHandler*, *OntologyProcessing* and *ServiceRepresentative*. The *ApplicationTemplate* class provides an abstraction for accessing the instance data based on application configurations. The *SemanticSearch* class extends *ApplicationTemplate* by including functions for accessing search-application-specific configurations, applying aggregation functions and result sorting. The *Recommender* class extends *ApplicationTemplate* by including functions for accessing recommender-application-specific configurations and manipulating recommendation properties and results. The *Mapping* class provides an abstraction for accessing mapping configuration and publishing the RDF data from a relational

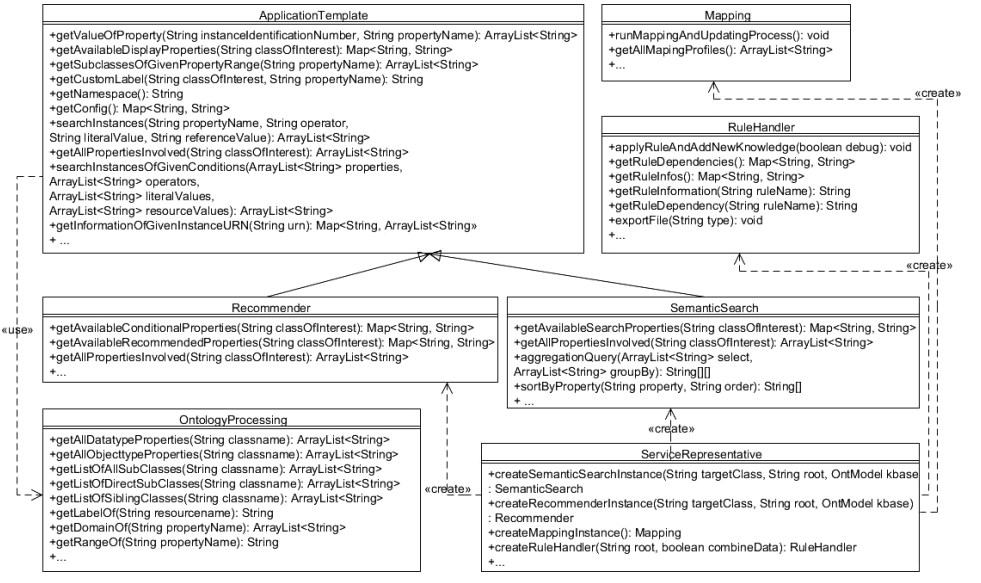


Fig. 11. Class diagram showing main classes of the OAM API.

database source based on the mapping configuration. The *RuleHandler* class is responsible for applying recommendation rules, adding new results to the knowledge base, obtaining dependency information between rules, and exporting rules. The *ServiceRepresentative* class acts as a class factory to create instance of other classes. The *OntologyProcessing* class is a utility class that provides an abstraction for accessing ontology data and handle variation of OWL syntax produced by different ontology editors.

Figure 12 shows a sequence diagram which exemplifies application development using the API in conducting a faceted search function. The application activates the *ServiceRepresentative* class for creating an *SemanticSearch* object named *searchObj*, given a class name whose instances will be searched, a directory path to the knowledge base and a data structure to store the instance data. The parameters are constructed as four *ArrayList <String>* objects: *properties*, *operators*, *literalValues* and *refValues*. The *properties* array contains a list of property names used in the search condition. The *operators* array contains a list of operators for each respective property name used in the search condition. The *literalValues* array contains a specified value when the respective property is data property or null when the property is object property. The *refValues* array contains a specified class name when the respective property is an object property or null when the property is a data property. Once all of the search conditions are added to the objects, the parameters are passed to a method of the *searchObj* object for retrieving the results. The method subsequently transforms the user's search conditions to a SPARQL query and returns the results formatted according to the search application configuration.

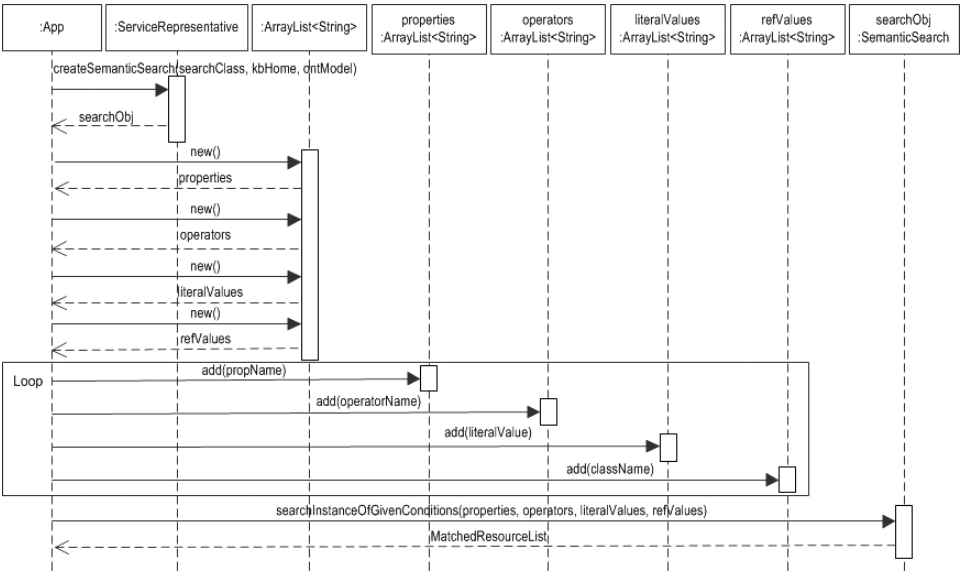


Fig. 12. Sequence diagram showing execution of a faceted search function using the OAM API.

4. Case Studies

This section presents two case studies that demonstrate practicality and applicability of the OAM framework in real-world settings. Two research projects that adopted the framework are used as case studies. In one project, OAM was used to support human activity recognition task in smart home domain [29]. Another project was development of an ontology-based thalassemia clinical support system for prevention and control of the disease [30]. Sections 4.1 and 4.2 provide descriptions of the case studies focusing on the roles and advantages of OAM framework in simplifying development in both projects. Section 4.3 provides some performance evaluation results that highlight benefit of the implementation-independent architecture of OAM.

4.1. *Ontology-based activity recognition engine in smart home*

4.1.1. *Scenario description*

Smart home has emerged as one of the mainstream approaches to support technology-driven independent living for elderly and disabled persons. The smart home concept [31], [32] is combined with several technologies, i.e. wireless sensor network, data communication, and security to produce ambient intelligence in the home. Activity recognition system plays an important role in realizing the information in the smart home environment and providing the relevant information back to the home application for supporting the independent living and ageing in place. Currently, activity recognition systems aim to capture what humans do on a daily basis [33], [34]. For example, knowing how long the home user sleeps at night [35] is considered relevant information for the homecare system. User context can be obtained based on diversity of sensors embedded into the objects in the home to obtain context-aware infrastructure information. Thus, one objective of this project is to develop a high-performance activity recognition framework based on ontology and the data collected from a diversity of sensors.

4.1.2. *Scenario implementation*

The context-aware activity recognition engine (CARE) architecture is proposed for automatic human activity recognition in the smart home domain. Figure 13 shows the high-level CARE architecture. In this architecture, ontology is used as a means to publish, access and analyse the large amount of data collected from a diversity of sensors. Specifically, ontology can provide the semantic information in the smart home about human activity and user's context. The ontology is mapped with the database storing data collected by a context sensor network (CSN) which obtains the surrounding information from the home and individual. A data manager is proposed to normalize and transform the data before storing it into the system repository. To process the data, ontology-based activity recognition (OBAR) is proposed for creating the knowledge base and activity model. Finally, the results of activity recognition and semantic information can be retrieved through the semantic

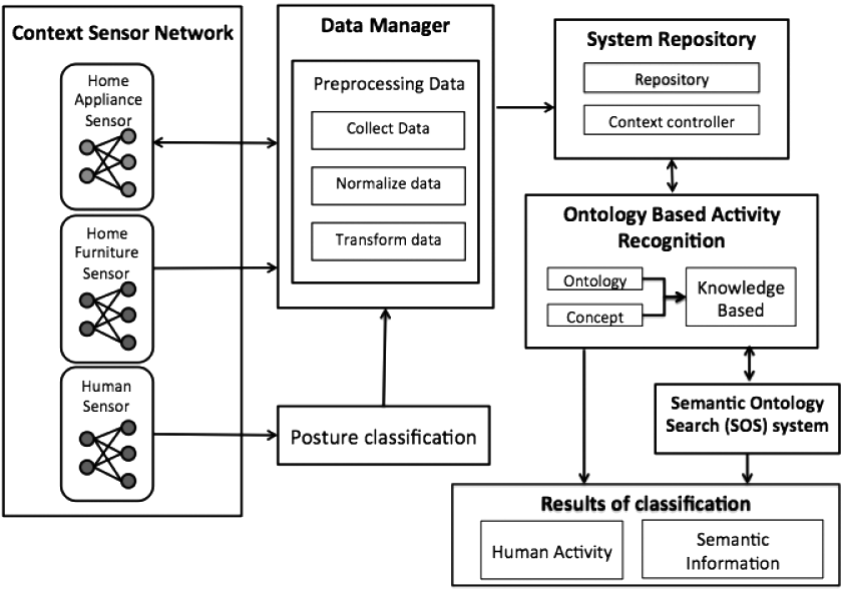


Fig. 13. Architecture of context-aware activity recognition engine.

ontology search system. Implementation details of the CARE architecture focusing on roles of OAM framework is discussed as follows.

Data Collection

For sensing data in this architecture, the proposed CSN is placed at the beginning of the architecture to collect the data from the home environment, including human information. There are three kinds of sensor networks in the CSN: Home Appliance Sensor Network, Home Furniture Sensor Network and Human Sensor Network.

- *Home Appliance Sensor Network.* To capture home appliance usage, a variety of sensors, such as power consumption sensors and water-flow sensors, are built into the smart home. Most electrical home appliances can be detected by measuring the change of electric current from the power consumption sensor. A water-flow sensor is also embedded in the smart home for monitoring the use of water fixtures such as "Sink," "Shower," and "Flushing." In our research, there are two protocols for sending the requested command to each sensor: ECHONET [36] and UPnP [37].
- *Home Furniture Sensor Network.* Furniture items in the home environment, such as "Sofa," "Chair," or "Bed", can be combined with the home appliance information to indicate the performance of specific activities. For example, a user tends to sit on a "Chair" when performing "Working on a computer" activity. In this sense, a pressure sensor is attached to the chair to detect whether or not a human is sitting on the chair. ZigBee protocol [37] is emulated for communication within the home furniture sensor network.

- *Human Sensor Network.* A human sensor network is used to observe human information such as the location of the individual. Infrared sensors are deployed in each room in the experimental environment to detect the human location. The current location of the user can give useful hints about which activities the individual is able to perform in the location.

Generally, posture classification and activity recognition are closely related research areas [38]. A range-based algorithm [39] which focuses on three human postures: "Standing," "Sitting," and "Lying down" is proposed. Three ultrasonic sensors were attached to the shoulder, hip, and knee to perform the range-based algorithm. The ranges between body parts along the y-axis were measured and used for determining the postures. For example, "Standing" and "Sitting" postures have different ranges between shoulder and knee, whereas a "Lying down" posture has little difference in range values between body parts. The information from the human body sensor is necessary to recognize the accurate activity results.

After the system obtains the data from the CSN and posture classification, the raw data are normalized by a supplied missing data function or an eliminating noisy data function. For example, an infrared sensor cannot recognize human location if the user is stationary or only moves a little. In this case, the supplied missing data function will retrieve the last location. A threshold technique is also adopted for filtering the noise in some circumstances. For example, even though an electrical device is turned off (but still plugged in), the power consumption sensor still perceives data from the electrical device. Thus, a lower boundary is set for removal of this kind of noise.

Ontology Development

The context-aware infrastructure ontology was designed to elicit the semantic context in the smart home. The superclass and subclass relationships were utilized. For example, the Electric appliance class and Furniture class are subclasses of the Object class. The Context class is a relevant class that relates location information and surrounding entities such as sensor, object, and human posture. For example, from Fig. 14, the "have_sensor" property in the Context class provides a relation to the

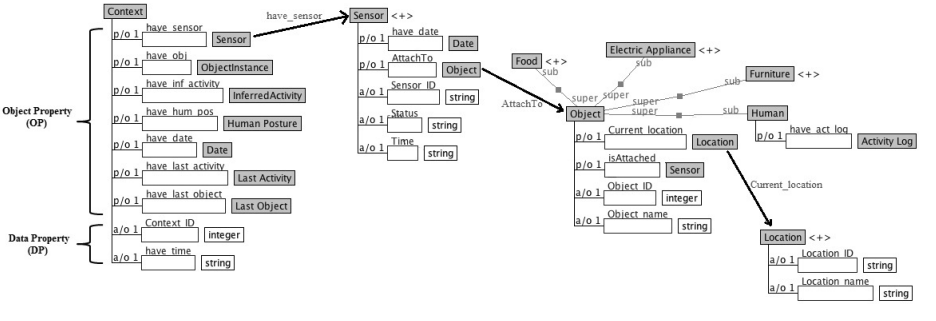


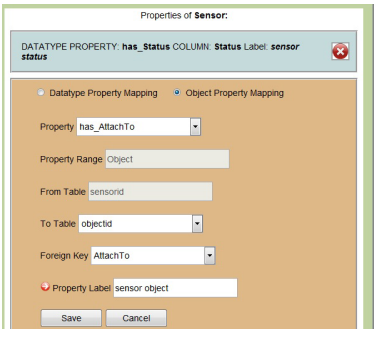
Fig. 14. Example of context-aware infrastructure ontology.

Sensor class, which is inherently linked to the Object class through the "AttachTo" property. The "Current_location" property in the Object class links to the Location class. Human posture information is also integrated into the Context class through the "have_hum_pos" property.

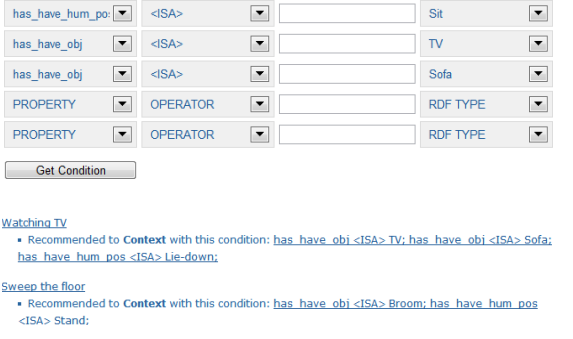
Data Processing

The system repository consists of a repository and a context controller. The repository is a relational database system. The context controller performs three main data-processing tasks.

- *Composing Data*. The data in the repository are formed to the user's context. We compose data every minute for one user's context. In the user's context, the system can perceive various kinds of semantic information such as object activated, sensor status, human posture, human location, and so on. OBAR will classify human activity based on these user contexts.
- *Mapping Data*. The ontology model perceives the data in the repository through this step. This task has duty to map between the properties and concepts in the ontology model and the data structure in the repository via OAM data mapping configuration as shown in Fig. 15(a). After this process, the system will generate the results as RDF data. The results of mapping will be stored in the smart home knowledge base for further processing.



(a) Mapping repository data to ontology



(b) Managing activity recognition rules

(c) Searching context data and activity recognition results using application template

Fig. 15. Example use of the OAM framework in ontology-based activity recognition in smart home domain.

- *Reprocessing Data.* Our proposed method also utilizes historical information in inferring human activity in the next classification. An external Java program built on top of OAM API was implemented for capturing temporal reasoning by collecting past classification results as an input for the next classification.

Ontology-Based Activity Recognition

In the activity recognition task, an ontology-based approach was adopted to exploit knowledge representation for context data modeling and to use logical reasoning to perform activity recognition. There are four factors to be considered in this research: object, human location, human posture and historical activity log. The following example indicates a rule for recognizing the "Wash dishes" activity in Description Logic syntax:

$$\begin{aligned} \textit{Wash dishes} \sqsubseteq\ & \textit{KitchenActivity} \\ & \sqcap\ \textit{use}(\textit{Object.Furniture}(\textit{Sink})) \\ & \sqcap\ \textit{Object.Human.Current\_location}(\textit{Kitchen}) \\ & \sqcap\ \textit{Human Posture}(\textit{Stand}) \\ & \sqcap\ \textit{LastActivity.Kitchen Activity}(\textit{Eating or drinking}) \end{aligned}$$

In this step, the OAM framework was used to create activity recognition rules using the provided rule editor and template as shown in Fig. 15(b). Subsequently, OAM transformed these rules to the syntax required by the rule-based reasoner. Based on the OAM rule template, the Inferred Activity class was created to link it with the inference results. At this stage, the smart home knowledge base has the rules for deciding upon the activity. If an input context reaches the system and is consistent with the rule linked with the instance of activities, the system will infer the activity as a result of recognition. The reasoner subsequently stores the new activity instance in the smart home knowledge base.

Knowledge Base Querying

The OAM search application template, i.e. semantic ontology search (SOS) system, is adopted on top of the CARE architecture for utilizing the knowledge from the OBAR. In this research, SOS is used for retrieving the history of semantic information in a smart home based on the ontology concepts. The system allows the users to set the search configuration based on the classes and properties in the ontology model and perform faceted search of data in the smart home knowledge base via SPARQL query templates. From a user viewpoint, the user can retrieve all semantic data including inferred activity results based on the class in the context-aware infrastructure ontology (object, sensor, or context) or the property in each class (date, time, or sensor status) as shown in Fig. 15(c). Evaluation of result accuracy is beyond the scope of this paper and is reported in [29].

4.2. *Ontology-based thalassemia clinical support system*

4.2.1. *Scenario description*

Thalassemia is the common genetic disease which can become global threat unless its spreading is effectively controlled. Prevention and control program for thalassemia has not been yet successful in Southeast Asian such as Thailand due to the lack of field-related experts and supporting facilities [40]. Since the thalassemia knowledge is highly complex and needs several specific areas of expertise, such as genetic analysis and interpretation of blood test, training sufficient number of thalassemic paramedical personnel cannot be done in short time. To expedite the process, a supporting system is necessary for assisting healthcare provider in making a decision or interpretation related to the disease. An ontological knowledge representation for thalassemia was developed to support related clinical services including thalassemia status diagnosis and prediction of thalassemia offspring.

4.2.2. *Scenario implementation*

Designing for future standardization and reusability, an ontology-based knowledge for thalassemia was constructed to support the clinical decision support system (CDSS) development. The domain ontology is used as a global schema that is mapped with the patient database which contains the patient health data lab results. The ontology is subsequently used together with the rule-based knowledge to process the data of examinees or patients for data analysis, e.g. interpretation of patient status and prediction of new cases.

Similar to the smart home case study, the architecture of the thalassemia CDSS involves ontology development, mapping of relational database data, i.e. patient database, to the ontology model, recommendation rule management, i.e. diagnosis and prediction rules, and knowledge base querying. Thus, the OAM framework helped to simplify deployment of the system in these steps. For brevity, this section only focuses on the role of OAM in this application while the deployment steps and details are not elaborated.

Figure 16 shows a layered architecture of the ontology-based thalassemia CDSS. A thalassemia ontology [30] was developed and aimed to provide support for four assisting services: (1) to inform individual's thalassemic status from the laboratorial test result, (2) to give probability of having a thalassemic child to at-risk couple (3) to provide a treatment option and alert to thalassemia patient and (4) to specify counselling options for counselling doctor. Currently only the first two services are implemented in the prototype system. The thalassemia ontology, diagnosis and interpretation rules, and a patient database can be provided as the input to the Computing layer, implemented using the OAM framework, which is responsible for generating a new copy of patient data and providing the analysis results for the paramedical personnel in the Service layer.

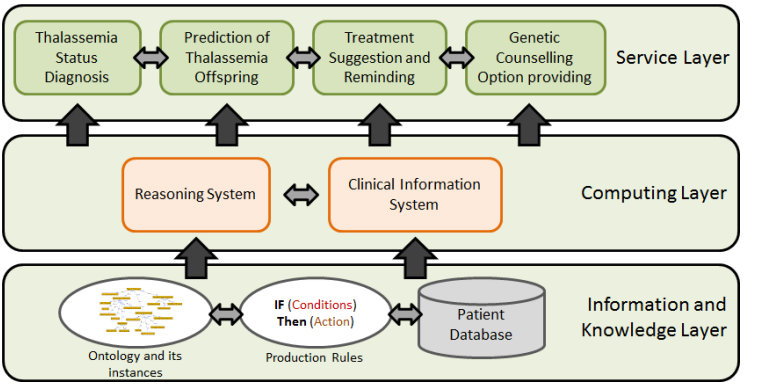


Fig. 16. A layered architecture of ontology-based thalassemia CDSS.

In developing the system, two deployment settings were considered: data analysis and interactive settings. In the data analysis setting, the purpose was to evaluate accuracy of the results produced by the diagnosis rules. Specifically, the study aimed to compare the CDSS diagnosis results with the existing actual physician diagnosis results. In supporting the study, the CDSS adopted the OAM database-ontology mapping function to generate the knowledge base of a clinical information system in RDF from based on the patient database and domain ontology. The OAM recommendation management function helped to simplify management of the diagnosis rules and allowed the knowledge engineer to focus on domain logics rather than rule language syntax. The knowledge base and recommendation rules were subsequently processed by the OAM recommendation processor which created the diagnosis results in the knowledge base. The semantic search application template was used for querying and assessing results in the knowledge base. No programming task was involved in this setting.

In the interactive setting, unlike the data analysis setting, a new diagnosis result is generated only when the user submits a new patient record to the system via a Web-based interface. In this setting, the OAM framework was adopted as an intermediate processing layer between Web application and patient database. Specifically, a Web service was additionally built on top of the OAM API. Adoption of the OAM framework in interactive setting, i.e. transaction-driven application, is beyond the scope of this paper and will be elaborated in future research.

4.3. *Performance evaluation*

This section provides evaluation results of OAM in terms of system response time in (1) publishing database data to the RDF knowledge base and (2) generating the recommendation results in the knowledge base. The purpose is to provide some indicators on scalability of the framework and to highlight benefit of the implementation-independent architecture adopted by OAM.

4.3.1. *Evaluation settings*

The system response time was measured in terms of data publishing time and data reasoning time. Data publishing time was measured from when the data is retrieved from the database and transformed to the RDF data based on the mapping configuration. Data reasoning time was measured from when the RDF data was read and processed by the recommendation processor component and the result was stored in the RDF data storage. In measuring the data reasoning time, the results of three different recommendation engine implementations were compared: Jena inference engine, Euler YAP Engine (EYE) and the SPARQL-based implementation of the OAM recommendation template. The tests were performed on a computer with the following specifications: Intel core i5 – 430M CPU with 8 GB DDR3 memories, using Java Development Kit version 1.6 with minimum memory size 1 GB and maximum memory size 4 GB. In each test, we warmed up the recommendation engines by loading data and perform reasoning until the execution time was stable. Then we measured the average system response time of three-time executions. The datasets and rules from the case studies were prepared as follows:

OBAR in Smart Home. In this case study, there were total of 33 rules designed for recognizing 13 targeted human activities. The test data were collected over a randomly selected one-day period which consists of the total of 351 contexts, i.e. almost six-hour of event data. Each context data is linked with the related collected data, such as human position, active objects, locations and related past objects and activities. The data is generated as five data sets in the size of 70, 140, 210, 280 and 351 contexts accordingly. The mapping configuration consists of mappings for nine classes, 13 object properties, seven data properties and 43 subclasses.

Thalassemia CDSS. In this case study, the total of 34 rules was created for diagnosis of 17 thalassemia disease and carrier types. The test data were extracted from the Siriraj hospital database entries of Thai patients who were examined for thalassemia between three years period. Each patient data consists of personal health data and lab results. Five data sets containing different numbers of patient data were prepared: 100, 500, 1000, 1500 and 2000 patients accordingly. The mapping configuration consists of mappings for 20 classes, 15 object properties, 24 data properties and 39 subclasses.

4.3.2. *Evaluation results*

OBAR in Smart Home. The summary of the data set (number of records, triples before and after reasoning) and overall performance results in terms of data publishing and reasoning time are shown in Table 1. The reasoning time is compared among three different recommendation engine implementations. Overall, the total response time for data publishing and reasoning is less than one minute for approximately six-hour event data and about 20 seconds for approximately one-hour event data. Jena inference engine has shown marginally better overall response time than the OAM's SPARQL-based implementation for data reasoning. Based on the

Table 1. Overall performance results of OBAR engine in smart home over the test data sets.

# of events	# of triples	# of triples after rec.	Data publishing time (sec)	Data reasoning time (sec)		
				Jena's	EYE	SPARQL
70	5,941	25,031	14.36	5.55	8.68	7.25
140	10,048	43,653	20.33	9.27	13.08	10.81
210	13,830	60,781	22.66	8.39	15.89	13.19
280	17,554	77,655	26.94	13.94	20.80	15.66
351	19,560	86,416	29.77	15.01	25.18	16.52

Table 2. Overall performance results of the thalassemia CDSS over the test data sets.

# of patients	# of triples	# of triples after rec.	Data publishing time (sec)	Data processing time (sec)		
				Jena's	EYE	SPARQL
100	8,238	63,724	20.88	10.76	21.74	12.76
500	31,438	291,702	46.37	80.13	105.91	37.26
1,000	60,438	576,473	75.30	266.38	238.87	61.35
1,500	89,438	861,472	103.37	548.94	425.44	83.94
2,000	118,438	1,146,892	131.04	772.60	567.88	112.28

results, recognizing daily human activities by OBAR engine can achieve a reasonable system response time using this framework.

Thalassemia CDSS. The summary of the data set and overall performance results in terms of data publishing and reasoning time are shown in Table 2. The reasoning time is compared among three different recommendation engine implementations. Overall, the total response time for publishing the data is approximately two minutes for 2,000 patient data. The total response time for data reasoning for 2,000 patient data is approximately less than ten minutes using Euler YAP engine and approximately less than two minutes using the OAM's SPARQL-based implementation. Based on the results, data reasoning in the thalassemia CDSS can achieve a reasonable system performance by choosing the appropriate implementation of the OAM recommendation engine.

The good performance of the SPARQL-based implementation is largely achieved by limited rule expressiveness of the OAM rule template. By limiting rule expressiveness, the customized SPARQL-based implementation is more efficient than the generic rule-based inference engines since less reasoning operations are performed.

5. Discussion

5.1. *Benefits and roles of the OAM framework in ontology-based application*

5.1.1. *Ontology-based data publishing and access*

The OAM framework publishes the RDF data that conform to a domain ontology. Thus, data from any database source that can be mapped to the domain ontology

can be merged for further publishing and processing. This allows the OAM-adopted system to support future data integration with other data sources that agree on the same domain ontology. In the case study of OBAR engine in smart home, a possible use case includes support for processing sensor-based data from different homes. Specifically, the OBAR engine can process any home's sensor database that can be mapped with the domain ontology and the resulted RDF data may be merged and shared with other smart home applications. In the thalassemia CDSS case study, a possible use case includes merging data from several hospital database sources for further data analysis.

5.1.2. *Abstraction*

In both case studies, the primary purpose of adopting the OAM Framework was to help in rapid prototyping and hypothesis testing. Both of the case studies focused on developing the domain knowledge and recommendation rules and evaluating the recommendation result accuracy. The OAM Framework is ideal for such use cases because it provides simplified data and application management functions to support data analysis application. In addition, using the OAM framework for ontology-based publishing of the existing database hides the complexity of relational database to RDF data mapping language. In addition, the recommendation rule template and editor hides the complexity of rule language syntax used by different rule-based reasoners. The application templates and API hide the complexity of SPARQL queries used in faceted searching over the knowledge base data. The simplification could contribute to promoting more adoption of the Semantic Web data standards and tools.

5.1.3. *Interoperability*

Although the current implementation relies largely on D2RQ and Jena framework, supporting additional implemented systems can be provided in future versions based on wrapper architecture. The OAM framework adopts the internal data representation that is implementation-independent. Thus, changing the underlying implemented systems will not require change on the user applications. For example, both of the case studies can benefit from improved performance of the implementation layer of the framework that would not require change in the user application layer.

In addition, the created knowledge base can be exported to the standard data formats for sharing and reuse by another system. Exported OAM knowledge base data typically includes ontology, instance and rule data as well as mapping and application configurations. In the thalassemia CDSS case study, this will allow the patient data and thalassemia knowledge, i.e. ontology and rules, to be reused and extended by other parties such as clinicians, researchers, hospitals, national statistics bureau, etc. In the case study of OBAR engine in smart home, a possible use case includes support for publishing home sensor databases that can be combined with

other sources [41]. Specifically, the shared knowledge base contains the user home context data that can be retrieved and mash-up with other data sources, such as mash-up with furniture and home device information. The OBAR knowledge, i.e. ontology and rules, can also be used and extended by another smart home application that wishes to utilize the knowledge.

5.2. *Limitations of the framework*

One of the limitations of the OAM framework is limited support for namespace management. Specifically, the created instance data will share the same namespace, e.g. a local namespace. Thus, the published knowledge base cannot reference the instance data outside the source database. This makes OAM has a limited support for Linked data application, which typically requires data referencing between different data sources Vocabulary mapping is also not currently supported by OAM. Vocabulary mapping is important for data integration and mash up. For example, one class or property in an ontology may be the same as another class or property that uses different name in another ontology. By providing vocabulary mapping facility on top of the application framework, the published knowledge base can be integrated or mashed up with other data sources that use different vocabularies. Thus, to allow for better support for Linked data, namespace and vocabulary management facilities should be additionally provided.

Although the templates, i.e. mapping, rule and query templates, were designed to support common uses of the generic languages, the templates have only limited features of the generic languages. For example, the current mapping configuration template does not support variable usage in join condition, which is important when joining a table to itself. In creating rules, rules that can not be represented using the recommendation template must be provided manually by the users. Using query template, some SPARQL features are not currently supported, such as optional, negation and subqueries. These support may be added in the future versions.

In addition, OAM shares the same limitations as the underlying systems, i.e. D2RQ, Jena's TDB [42]. For example, D2RQ provides no support for update or multiple database integration facility to the published data [43]. These features may be additionally provided by the application framework. In terms of system performance, based on the Berlin SPARQL Benchmark results [44], TDB is not as scalable as some triplestores or relational database systems. Future research will investigate different implementations of RDF data store by means of the wrapper architecture.

5.3. *Conclusion and future work*

With the ongoing Semantic Web initiative, development tools that allows for rapid prototyping of ontology-based applications are needed. This paper introduces an application framework designed for ontology-based Semantic Web applications that focuses on providing reusable and configurable data and application templates.

Adopting the templates allows the users to create the applications without programming skills required. Three forms of templates, which are specific forms of existing languages, are proposed: database to ontology mapping configuration, recommendation rule and application templates. We describe two case studies that adopted the framework and how the framework was used in simplifying development in both projects. In addition, we provide some performance evaluation results to show that, by limiting expressiveness of the rule language, a specialized form of recommendation processor can be developed for more efficient performance. Some advantages and limitations of the application framework in ontology-based applications are also discussed.

Some future development includes adding support for more application templates such as decision support system and visualization applications. Future investigation will focus on improving system performance, compatibility with various Semantic Web data standards and tools and improved support for Linked Data applications.

Acknowledgments

The authors would like to acknowledge the financial supports from the Service Informatics and Young Scientist and Technologist Programmes, National Science and Technology Development Agency (NSTDA), Thailand. All contributors to the software tool development and adoption are gratefully acknowledged.

References

1. C. Bizer, T. Heath and T. Berners-Lee, Linked data — the story so far, *Int. J. Semant. Web Inf. Syst.* **5**(3) (2009) 1–22.
2. B. Heitmann, K. Sheila, H. Conor and S. Decker, Implementing semantic web applications: Reference architecture and challenges, in *Proc. 5th Int. Workshop on Semantic Web-Enabled Software Engineering*, 2009.
3. M. Fayad and D. C. Schmidt, Object-oriented application frameworks, *Commun. ACM* **40**(10) (1997) 32–38.
4. E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software* (Pearson Education, 1994).
5. H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann and S. Hübner, Ontology-based integration of information — a survey of existing approaches, in *Proc. Workshop on Ontologies and Information Sharing*, 2001, pp. 108–117.
6. D. Calvanese, G. De Giacomo and M. Lenzerini, Description logics for information integration, *Computational Logic: Logic Programming and Beyond. LNCS* **2408** (2002) 41–60.
7. A. Schultz, A. Matteini, R. Isele, P. Mendes, C. Bizer and C. Becker, LDIF — a framework for large-scale linked data integration, in *Proc. 21st International World Wide Web Conference*, 2012.
8. B. Haslhofer, *A Web-Based Mapping Technique for Establishing Metadata Interoperability*, University of Vienna, 2008.
9. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi and D. F. Savo, The MASTRO system for ontology-based data access, *Semant. Web* **2**(1) (2011) 43–53.

10. E. Kharlamov *et al.*, Optique: Towards OBDA systems for industry, in *Proc. ESWC (Satellite Events)*, 2013, pp. 125–140.
11. J. J. Carroll, D. Reynolds, I. Dickinson, A. Seaborne, C. Dollin and K. Wilkinson, Jena?: Implementing the Semantic Web recommendations, in *Proc. of the 13th international World Wide Web Conference on Alternate Track Papers & Posters*, 2004, pp. 74–83.
12. R. García-Castro, A. Gómez-Pérez, Ó. Muñoz-García and L. J. B. Nixon, Towards a component-based framework for developing Semantic Web applications, in *Proc. 3rd Asian Semantic Web Conference*, 2008, pp. 197–211.
13. E. Oren, A. Haller, C. Mesnage, M. Hauswirth, B. Heitmann and S. Decker, A flexible integration framework for Semantic Web 2.0 applications, *IEEE Softw.* **24**(5) (2007) 64–71.
14. M. Hausenblas, Exploiting linked data to build web applications, *IEEE Internet Comput.* **13**(4) (2009) 68–73.
15. S. Auer *et al.*, Managing the life-cycle of linked data with the LOD2 stack, in *Proc. 11th Int. Conf. on the Semantic Web — Volume Part II*, 2012, pp. 1–16.
16. T. Kurz, S. Schaffert and T. Burger, LMF: A framework for linked media, in *Proc. Workshop on Multimedia on the Web (MMWeb)*, 2011, pp. 16–20.
17. M. Buranarach, Y. Thein and T. Supnithi, A community-driven approach to development of an ontology-based application management framework, *Semantic Technology, LNCS*, **7774** (2013) 306–312.
18. C. Bizer and A. Seaborne, D2RQ-treating non-RDF databases as virtual RDF graphs, in *Proc. of the 3rd Int. Semantic Web Conf.*, 2004.
19. Apache Jena — Reasoners and rule engines Jena inference support. [Online]. Available: http://jena.apache.org/documentation/inference [Jan. 4, 2015].
20. K. Kozaki, Y. Hayashi, M. Sasajima, S. Tarumi and R. Mizoguchi, Understanding Semantic Web applications, in *Proc. of the 3rd Asian Semantic Web Conf.*, 2008, pp. 524–539.
21. C. Bizer, D2R MAP — a database to RDF mapping language, in *Proc. of the 12th International World Wide Web Conf.*, 2003.
22. T. C. Will, A. Srinivasan, I. Im and Y. B. Wu, Search personalization: Knowledge-based recommendation in digital libraries, in *Proc. AMCIS 2009*, 2009, p. 728.
23. M. Kifer and H. Boley, *RIF Overview* (Second Edition), W3C Working Group Note, 2013. [Online]. Available: http://www.w3.org/TR/rif-overview/ [Jan. 4, 2015].
24. T. Berners-Lee and D. Connolly, Notation3 (N3): A readable RDF syntax, W3C Team Submission, 2011. [Online]. Available: http://www.w3.org/TeamSubmission/n3/ [Jan. 4, 2015].
25. Euler Yet Another Proof Engine. [Online]. Available: http://eulersharp.sourceforge.net/ [Jan. 4, 2015].
26. S. Hawke, I. Herman, B. Parsia and A. Seaborne, SPARQL 1.1 Entailment Regimes, W3C Recommendation, 2013. [Online]. Available: http://www.w3.org/TR/sparql11-entailment/ [Jan. 4, 2015].
27. H. Knublauch, J. A. Hendler and K. Idehen, SPIN — Overview and Motivation, W3C Member Submission, 2011. [Online]. Available: http://www.w3.org/Submission/spin-overview/ [Jan. 4, 2015].
28. M. Hearst, Design recommendations for hierarchical faceted search interfaces, in *Proc. ACM SIGIR Workshop on Faceted Search*, 2006.
29. K. Wongpatikaseree, A. O. Lim, M. Ikeda, and Y. Tan, High performance activity recognition framework for ambient assisted living in the home network environment, *IEICE Trans.* **97-B**(9) (2014) 1766–1778.
30. A. Assawamakin, N. Chalortham, T. Ruangrajitpakorn, C. Limwongse, T. Supnithi and S. Tongsima, A development of knowledge representation for thalassemia prevention and

control program, in *Proc. 7th Int. Conf. on Natural Language Processing and Knowledge Engineering*, 2011, pp. 190–193.
31. M. Chan, D. Estève, C. Escriba and E. Campo, A review of smart homes — present state and future challenges, *Comput. Methods Programs Biomed.* **91**(1) (2008) 55–81.
32. L. C. De Silva, C. Morikawa and I. M. Petra, State of the art of smart homes, *Eng. Appl. Artif. Intell.* **25**(7) (2012) 1313–1321.
33. J. B. J. Bussmann, W. L. J. Martens, J. H. M. Tulen, F. C. Schasfoort, H. J. G. Berg-Emons, and H. J. Stam, Measuring daily behavior using ambulatory accelerometry: The activity monitor, *Behav. Res. Methods, Instruments, Comput.* **33**(3) (2001) 349–356.
34. S.-W. Lee and K. Mase, Activity and location recognition using wearable sensors, *Pervasive Comput. IEEE* **1**(3) (2002) 24–32.
35. A. Gaddam, S. C. Mukhopadhyay and G. S. Gupta, Integrating a bed sensor in a smart home monitoring system, in *Proc. IEEE Instrumentation and Measurement Technology*, 2008, pp. 518–521.
36. S. Matsumoto, Echonet: A home network standard, *Pervasive Comput. IEEE* **9**(3) (2010) 88–92.
37. K.-S. Kim, C. Park, K.-S. Seo, I.-Y. Chung and J. Lee, ZigBee and the UPnP expansion for home network electrical appliance control on the Internet, in *Proc. 9th Int. Conf. on Advanced Communication Technology*, 2006, pp. 1857–1860.
38. M.-W. Lee, A. Khan and T.-S. Kim, A single tri-axial accelerometer-based real-time personal life log system capable of human activity recognition and exercise information generation, *Pers. Ubiquitous Comput.* **15**(8) (2011) 887–898.
39. K. Wongpatikaseree, A. O. Lim, Y. Tan and H. Kanai, Range-based algorithm for posture classification and fall-down detection in smart homecare system, in *Proc. IEEE 1st Global Conference on Consumer Electronics*, 2012, pp. 243–247.
40. S. Fucharoen and P. Winichagoon, Prevention and control for thalassemia in Asia, *Asian Biomed.* 1(1) (2007) 1–6.
41. D. Le-Phuoc, H. N. M. Quoc, J. X. Parreira and M. Hauswirth, The linked sensor middleware–connecting the real world and the semantic web, in *Proc. of the Semantic Web Challenge*, 2011.
42. The Apache Software Foundation, Apache Jena — TDB. [Online]. Available: http://jena.apache.org/documentation/tdb/ [Jan. 4, 2015].
43. C. Bizer and R. Cyganiak, D2RQ — Lessons Learned, Position paper for the W3C Workshop on RDF Access to Relational Databases, 2007. [Online]. Available: http://www.w3.org/2007/03/RdfRDB/papers/d2rq-positionpaper/ [Jan. 4, 2015].
44. C. Bizer and A. Schultz, The Berlin SPARQL Benchmark, *Int. J. Semant. Web Inf. Syst.* **5**(2) (2009) 1–24.