Design and Implementation of a Rule-based Recommender Application Framework for the Semantic Web Data

Thanyalak Rattanasawad[1], Marut Buranarach[2], Ye Myat Thein[2], Thepchai Supnithi[2], and Kanda Runapongsa Saikaew[1]

[1] Department of Computer Engineering, Faculty of Engineering, Khon Kaen University, Khon Kaen, Thailand
thanyalak.rattanasawad@gmail.com, krunapon@kku.ac.th
[2] Language and Semantic Technology Laboratory
National Electronics and Computer Technology Center (NECTEC), Pathumthani, Thailand
{marut.bur, thepchai.sup}@nectec.or.th, yemyatthein@gmail.com

Abstract. This paper describes the design and implementation of a recommender application framework which aims to simplify development of ontology-based recommender applications over the Semantic Web data. 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, our application framework introduces a recommendation template that is a specific form of rule language that provides high-level abstraction in generating recommendations. Recommendation rules can be created based on the template using recommendation editor to hide complexity of rule language syntax. The framework proposes two implementation approaches for generating recommendation results based on the recommendation rules: rule-based reasoner and SPARQL-based implementation. The preliminary evaluation results over a data set in public health domain showed more efficient execution time using the SPARQL-based implementation approach. The results suggested that, by limiting expressiveness of recommendation rules, a specialized form of recommendation processor can be developed for more efficient system performance.

Keywords: Semantic Web application framework, ontology application framework, rule-based recommender system framework

1 Introduction

Although creation of the Semantic Web data rapidly grows, e.g. the Linked data initiatives [1], applications and uses of the data are relatively limited. This is partly due to high learning curve and efforts demanded in building Semantic Web and ontology-based applications. Recommender system is a type of system that generates meaningful recommendations to support user's decision. Recommender system development 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, development tools should allow application developers to focus more on domain problems and knowledge rather than implementation details.

Our work proposes an application framework aimed to simplify development of recommendation systems that apply the knowledge-based analysis technique [2] over the Semantic Web data. A typical application of the technique includes modeling a user's profile information and resource properties based on ontology. Then, a recommendation of list of resources recommended for the user can be generated based on recommendation rules externally defined by domain expert users. Our framework introduces a recommendation template that is a specific form of rule language that provides high-level abstraction in generating such a recommendation. Recommendation rules can be created based on the template using recommendation editor to hide complexity of rule language syntax. Subsequently, the template can be processed by a recommendation processor to produce recommendation results.

This paper describes the design and implementation of the recommender application framework developed as a part of the Ontology Application Management (OAM) Framework [3], which is an application framework aimed to simplify development of ontology-based applications over the Semantic Web data. Design of the framework focuses on three main principles: abstraction, extensibility, and interoperability. It proposes a generalized recommendation template for ontology-based recommender applications. We also present two implementation approaches of recommendation processor to execute the recommendation rules and evaluate the performance over a data set in public health domain. The evaluation results showed that, by limiting expressiveness of recommendation rules, a specialized form of recommendation processor can be developed for more efficient system performance.

2 OAM Framework

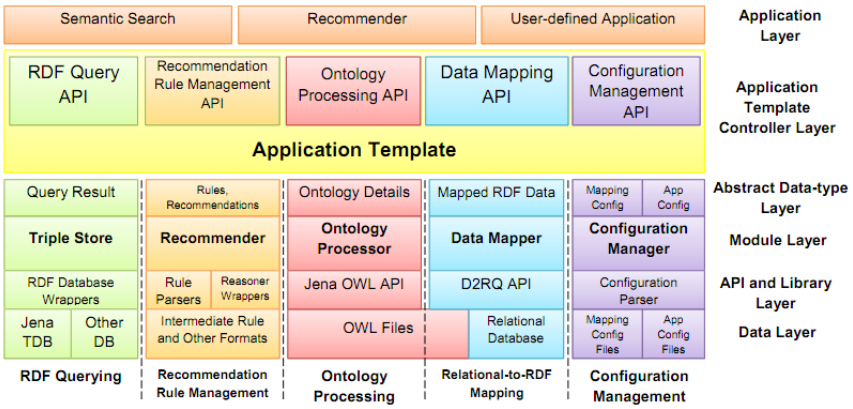

Fig. 1 Architecture of OAM framework

Ontology-based Application Management (OAM) framework [3] is a framework to simplify development of semantic web applications focusing on semantic search and

recommender systems. It enables the user to create customized applications based on provided application templates without high learning curve. The framework was developed in Java programming language on top of existing tools: Jena framework[1] for processing RDF-based data and D2RQ[2] for mapping data from relational databases to RDF database. The layered architecture of the OAM framework is shown in Fig. 1. The framework provides Application Template, the base template for creating semantic web applications. It is a collaboration of application modules for processing each type of data: Triple Store for storing and querying RDF data, Recommender for creating recommendations and managing recommendation rules, Ontology Processor for processing details of ontology, Data Mapper for mapping data from relational database to RDF database, and Configuration Manager for managing configurations of the data mapping and other applications. This paper focuses on the Recommender module of the OAM framework.

3 Recommendation Template

The recommender application framework focuses on simplifying creation and management of recommendation rules. It provides a recommendation rule management tool that supports two processes: 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 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 and hides complexity of the rule syntax to be processed by reasoning engine.

In this framework, recommendation rule can be created based on a recommendation template, whose structure 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> <oper-

[1] http://jena.apache.org/
[2] http://d2rq.org/

ator> <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".

Fig. 2 illustrates elements of the recommendation rule and linked condition sets in JSON format. Condition sets are illustrated in Fig. 3. The user can use a provided rule editor to create recommendation rules based on the template that hides the complexity of the created rule syntax.

```
"ClearancePromotion": {
  recOfRule: "DiscountedProducts",
  recToRule: "FrequentlyVisitingCustomers",
  recProperty: "http://ex.org/suggestedProducts" }
```

Fig. 2 Elements of recommendation rule and linked condition sets

```
"DiscountedProducts": {
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"]]}

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

Fig. 3 Matching Condition Sets

4 Design and Implementation of the Recommender Application Framework

The 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 data-

bases. Interoperability is achieved by allowing exports to the standard rule formats to enable interchanging and integration with other rule-based systems.

4.1 System Architecture

Fig. 4 illustrates the system architecture of the recommender module. The follows describe details of the related modules of recommendation data management.

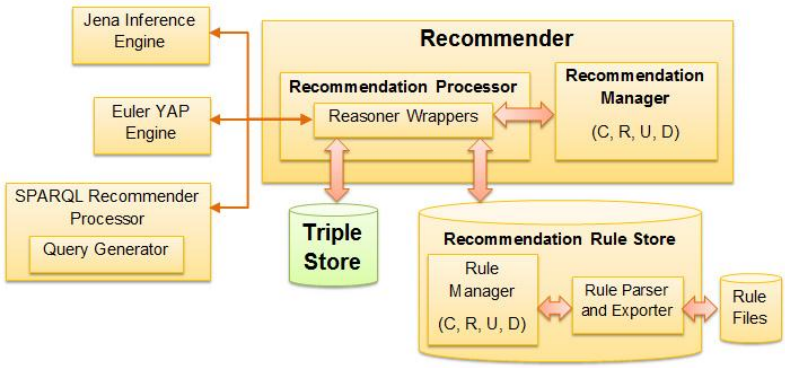

Fig. 4 Architecture of the OAM recommender module

- **Triple Store:** The module which 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 [4], and a format of supported reasoner, such as Jena rule [5] and Notation3 [6].
- **Recommendation Processor:** The sub-module which provides common interfaces and encapsulates algorithms of recommendation creation, and recommendation results generation.

4.2 Implementations of Recommendation Processor

Recommendations are created and linked with the related resources by passing recommendation rules and data to the recommendation processor. 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 [5] and Euler YAP Engine [7]. 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. Fig. 5 shows an example of the recommendation rule created in Jena rule syntax.

```
@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 :hasInstances ?recOf)

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

Fig. 5 Jena rule syntax of the recommendation rule

- **SPARQL-based approach:** This approach adopts a SPARQL engine as the recommendation processor [8][9], and generates recommendation rules by means of SPARQL queries and updates. Fig. 6 shows an example of SPARQL update syntax for generating recommendations and results.

```
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. 6 SPARQL update syntax of the recommendation rule

5. Performance Evaluation

The performance of the framework implementations were evaluated in supporting the Thalassemia clinical decision support system (CDSS) research [10]. The total of 34 rules was created based on the template to enable 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. Each patient data consists of personal health data, lab and DNA test results. Five data sets containing different sizes of patient data were prepared: 100, 500, 1000, 1500 and 2000 patients. Data processing time was measured from when the RDF/XML file was read and processed

by the recommendation processor and stored in the RDF data storage. Wrappers for three different implementations of the recommendation processor, Jena inference engine, Euler YAP Engine (EYE) and the SPARQL-based implementation were deployed and compared in terms of system response time. The tests were performed on a computer with the following specifications: Intel core i5 – 430M CPU with 8 GB DDR3 memories, using JDK version 1.6 with minimum memory size 1 GB and maximum memory size 4GB.

The detailed comparison of total processing time for different implementations is shown in Fig. 7. The total response time for data processing and analysis for 2,000 patient data, which consist of 0.12 million triples and 1.14 million triples before and after recommendation respectively, is approximately under 14 minutes using Jena's, under ten minutes using EYE and under two minutes using the SPARQL-based implementation. Based on the results, the Thalassemia CDSS can achieve a reasonable system performance by using the SPARQL-based implementation.

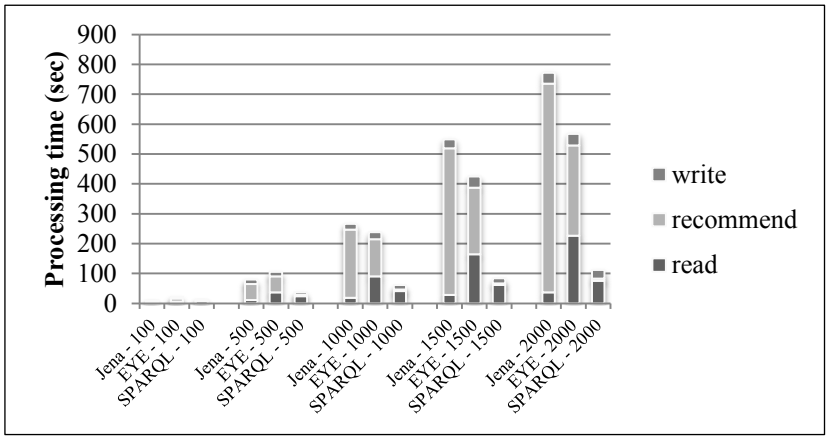

Fig. 7 Results of data processing time in Thalassemia CDSS using different implementations

The good performance of the SPARQL-based implementation is largely achieved by limiting rule expressiveness. Currently, it can only support rules that generate results which do not fire another rule. By limiting rule expressiveness of the recommendation template, the SPARQL-based implementation is more efficient than rule-based inference engine implementation since less reasoning operations are performed.

6. Conclusion

In this paper, we presented design and implementation of a rule-based recommender application framework for the Semantic Web data. The framework is different from existing Semantic Web application framework in that it simplifies the development of recommender applications by providing a generalized recommendation template that can be used in ontology-based recommender applications. The template can be managed and processed by specialized recommendation editor and processor. The design of the framework also focuses on extensibility and interoperability to allow it to be

independent of underlying implemented systems. An evaluation study was conducted by comparing performance of different implementations of the recommendation processor using a data set in public health domain.

Our planned future work includes improving design of the recommendation template to support more expressiveness and functionality. We also plan to improve the SPARQL-based implementation to support more expressive recommendation rules.

Acknowledgement

The financial support from Young Scientist and Technologist Programme, NSTDA (YSTP: SP-56-NT03) is gratefully acknowledged.

References

1. Bizer, C., Heath, T., Berners-Lee, T.: Linked Data - The Story So Far. Int. J. Semant. Web Inf. Syst. 5, 1–22 (2009).

2. Burke, R.: Knowledge-Based Recommender Systems. Encycl. Libr. Inf. Syst. 69, (2000).

3. Buranarach, M., Thein, Y.M., Supnithi, T.: A Community-Driven Approach to Development of an Ontology-Based Application Management Framework. Joint International Semantic Technology Conference (2012).

4. Kifer, M., Boley, H.: RIF Overview (Second Edition), http://www.w3.org/TR/rif-overview/.

5. The Apache Software Foundation: Apache Jena - Reasoners and rule engines Jena inference support, http://jena.apache.org/documentation/inference.

6. Berners-Lee, T., Connolly, D.: Notation3 (N3): A readable RDF syntax, http://www.w3.org/TeamSubmission/n3/.

7. Roo, J. De: Euler Yet Another Proof Engine, http://eulersharp.sourceforge.net/.

8. Hawke, S., Herman, I., Parsia, B., Seaborne, A.: SPARQL 1.1 Entailment Regimes, http://www.w3.org/TR/sparql11-entailment/.

9. Knublauch, H., Hendler, J.A., Idehen, K.: SPIN - Overview and Motivation, http://www.w3.org/Submission/spin-overview/.

10. Assawamakin, A., Chalortham, N., Ruangrajitpakorn, T., Limwongse, C., Supnithi, T., Tongsima, S.: A development of knowledge representation for thalassemia prevention and control program. Natural Language Processing and Knowledge Engineering (NLP-KE), 2011 7th International Conference on. pp. 190–193 (2011).