Rulebase Integration for eCollaboration

Duong, D.D.; Boley, Harold; Le, T.T.T.; Bhavsar, V.C.

e-Networks in an Increasingly Volatile World [Proceedings], 2006

National Research Council Canada
Conseil national de recherches Canada

Institute for Information Technology
Institut de technologie de l'information

NRC·CNRC

*Rulebase Integration for eCollaboration**

Duong, D.D., Boley, H., Le, T.T.T., and Bhavsar, V.C
August 28-31, 2006

* Proceedings of the 11th International Workshop on Telework. Fredericton, NB. August 28-31, 2006. NRC 49311. ISBN 1-55131-107-0. pp. 2-17. 2006.

Canada

Rulebase Integration for eCollaboration

DOAN Dai Duong[1] , Harold BOLEY[2], LE Thi Thu Thuy[1] and Virendrakumar C. BHAVSAR[1]
1 Faculty of Computer Science, University of New Brunswick,
Fredericton, New Brunswick, Canada
{Duong_Dai.Doan, Thuy_Thi_Thu.Le, bhavsar} AT unb.ca
2 Institute for Information Technology e-Business,
National Research Council of Canada,
Fredericton, New Brunswick, Canada
harold.boley AT nrc-cnrc.gc.ca

Abstract. This paper provides a classification of rulebase integration for eCollaboration. A variety of conflicts between rulebases are identified and guidelines for conflict resolutions are suggested. Based on the classification, a framework for rulebase integration is proposed containing two different integration approaches, namely interoperation and interchange. The problem of semantics-preserving rulebase transformation is discussed, and a solution is given. Function serialization and representation during transformation are also discussed and represented in terms of Functional RuleML.

Keywords. Rulebase integration, homomorphism, lossy transformation, conflict resolution.

1. Introduction

This paper describes foundational work in semantic information integration central to the cluster of Semantic Web projects at UNB and NRC Fredericton, which are being applied in eBusiness [2], eLearning [4] and eCollaboration [12] scenarios. We show how to define the objects and vocabularies of eCollaboration (e.g., merchandise, services) by rules (including taxonomies). Usually those rules differ, syntactically, semantically, and pragmatically, between (groups of) participants of Web-based collaborations. Therefore rulebase integration techniques are needed, as classified in the current paper.

The Semantic Web community has its focus on semantic information sharing and reuse. Underlying many Semantic Web applications are rule-based systems. However, the techniques for integration of distributed rule-based systems are still limited. In April 2005, the first workshop of Rule Interoperability [21] held by W3C, therefore, was a kick-off event for the development of interoperability between heterogeneous rule-based systems. Recalling the work on databases integration, it should be noted that this problem has been a

long-standing challenge in the Database Community. However, the issue of rulebase integration is even more complex. A rule contains a head and a body, which are a consequent (conclusion) and an antecedent (condition), respectively. When the body is empty (i.e., no condition), the rule becomes a fact. Database integration [1, 8, 18] mostly deals with such facts in the form of relational tables. Therefore, databases integration can be regarded as a special case of rulebase integration.

Although more work has been done in database integration, there is a growing body of research focusing on rulebase integration. Three examples follow. In [6], early literature on modularity in logic programming was surveyed, where a program (rulebase) can be regarded as a combination of separate and independent components (modules). The classification in their paper is based on two main streams. An integrated program can either be formed based on the construction of an algebra, which links operators of subprograms, or be defined as linguistic extensions (abstraction) of Horn clauses. In [10], the authors gave some reasons for the need of rule interoperability and discussed some basic requirements that a language for interoperability must satisfy for broader use. In [14], the authors strived to use SWRL as a platform language for rule and ontology integration. They provided a homogeneous rule and ontology integration environment, where third party rule engines can be plugged-in. However, in their system, the interoperability was done on the API level. The extension and the interaction of the systems with other non-API rule-based systems, therefore, need to be further invested.

In this paper, we propose a classification, framework, and issues of rulebase integration as needed for eCollaboration. This will provide a better understanding of, and a foundation for further development in, rulebase integration. Along with the classification, we discuss the difficulties of handling conflicts between rulebases and suggest various solutions for them. Based on the classification, we propose a general framework for rulebase integration, which includes both *rulebase interoperation* and *rulebase interchange* in the sense of the distinction proposed by Allen [15] and the W3C RIF group [17]. In the interchange approach, we use homomorphisms to preserve the semantics of rulebases before and after transformation. When transforming a rulebase from a source language to a target language, sometimes we need to split this rulebase into several parts and interchange the maximum subset of the rulebase that is compatible with the target rule engine. Another key topic of rulebase integration is how to represent constructors, user-defined, and built-in functions in rulebases while transporting them on the web. Our solution is to serialize these functions by Functional RuleML [4].

Section 2 classifies rulebase integration. Section 3 presents a framework for both rulebase interoperation and rulebase interchange. Section 4 discusses issues of rulebase interchange. We emphasize the need for preserving the semantics of a rulebase before and after transformation as well as suggest a technique for serializing functions on rulebases. Results on experiments for transforming between rule languages are also given.

2. A Classification of Rulebase Integration

We classify rulebase integration based on two dimensions: surface syntax and expressive(ness) fragment. Heterogeneous rulebases can be serialized in different languages and use various fragments of expressiveness. The following diagram (Figure 1) shows our principal classification of rulebase integration.

In the top-most level, the integration is divided based on the differences of incoming rule languages (i.e., a fixed surface syntax vs. different surface syntaxes). Since rulebases of heterogeneous rule languages can be transformed into a canonical form, the work of rulebase integration involving different surface syntaxes can be achieved by that of rulebase integration using a fixed surface syntax after a syntactic conversion is applied for these different surface syntaxes. The next sections present the classification of rulebase integration in detail.

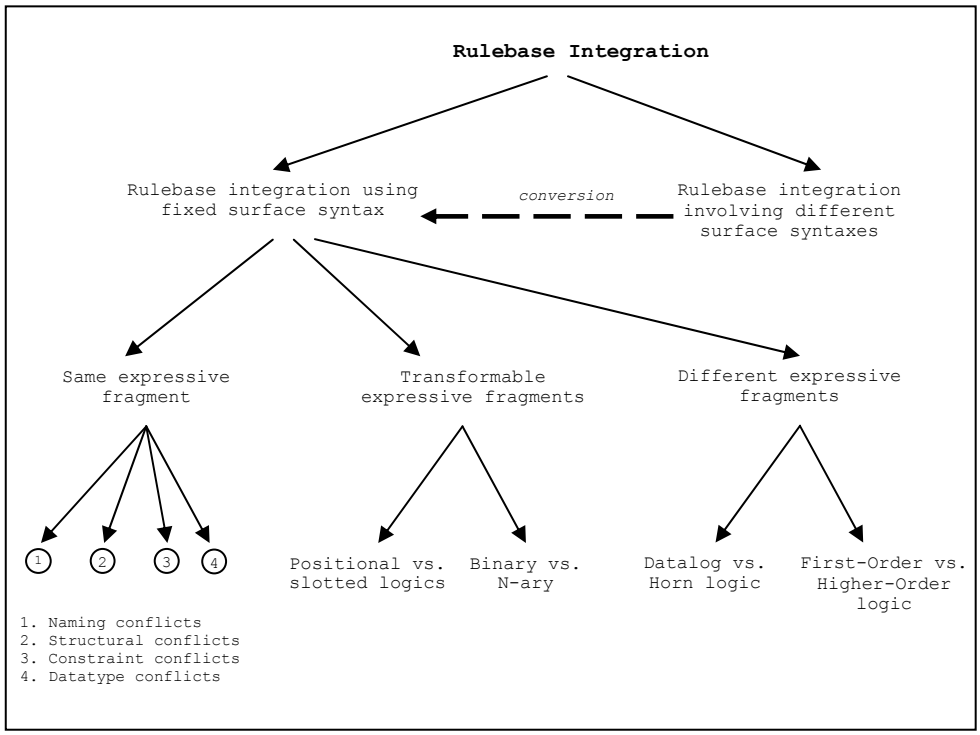

Figure 1. Classification of rulebase integration

2.1. Integration of Rulebase Using Fixed Surface Syntax

Integration of rulebases of fixed surface syntax can be subdivided into three sub-scenarios: Integration of rulebases having the same expressive fragment, integration of rulebases having transformable expressive fragments, and integration of rulebases having different expressive fragments.

2.1.1. Integration of Rulebases Having the Same Expressive Fragment

Rulebases, even serialized in the same surface syntax and having the same expressiveness, are often heterogeneous. The reason is that different rulebase providers have different perspectives about an issue. Therefore, the key issue of rulebase integration is to reconcile conflicts between various rulebases. Conflicts are classified into four main types: *Naming conflicts, structural conflicts, datatype conflicts* and *constraint conflicts.* In this paper, we only discuss naming conflicts, structural conflicts, and datatype conflicts. For work on constraint conflicts and their resolution see [7, 8].

a. Naming Conflicts
There are two main different types of naming conflicts, namely *synonyms* and *homonyms*.

Synonyms: When two different terms (i.e., relations in rules) refer to the same real world object or concept, they are known as synonyms. For example, a `merchandise` in an eBusiness application is declared by a rulebase provider R_1 as follows:

RB_1

```
merchandise(X) :- provider(Y,X), warehouse(Y,Z).
provider("Compact Corp.", "Printer").
warehouse("Compact Corp.","Boston").
```

This Datalog rulebase contains a rule and two facts. The rule specifies that `merchandise X` is provided by `provider Y` from the `warehouse Z`. Here uppercase letters (e.g., `X, Y, Z`) represent variables. On the right-hand side of the symbol `":-"`, (i.e., `provider(Y,X), warehouse(Y,Z))` is the body (condition) while on the left-hand side (i.e., `merchandise(X)`) is the head (conclusion) of the rule. The condition can either be an atom or a conjunction of atoms while the conclusion can only be an atom. However, from the point of view of another rulebase provider R_2, the above `merchandise` can be formalized as follows:

RB_2

```
item(X) :- supplier(Y,X), store(Y,Z).
supplier("Compact Corp.", "Printer").
store("Compact Corp.", "Boston").
```

Intuitively, these two different rulebases, RB_1 and RB_2, have the same semantics. The only difference is that the relations of RB_1 (`merchandise, provider,`

warehouse) and RB_2 (item, supplier, store) are expressed in different terms which, however, are synonyms. However, since rule interpreters are not as intelligent as humans, these two rulebases are considered different. Usually, a term dictionary would be provided for one to one transformation between relation names.

Relation subsumption

Let us consider two relations, namely P and P′, of two different clauses, L and L′ respectively.

Defintion 1

Let P and P′ be two relations. P and P′ are 'subsumption- interoperable' if they are on the same path in a relation hierarchy diagram (relation 'taxonomy'), i.e., a relation node is a parent (child) node of another relation node.

For example:

P = merchandise
P′= product

and based on information in RDFS (Figure 2), we have merchandise is subClassOf of product, which could be written as the following second-order facts: subClassOf(merchandise, product) . We can then conclude that P and P′ are subsumption-interoperable.

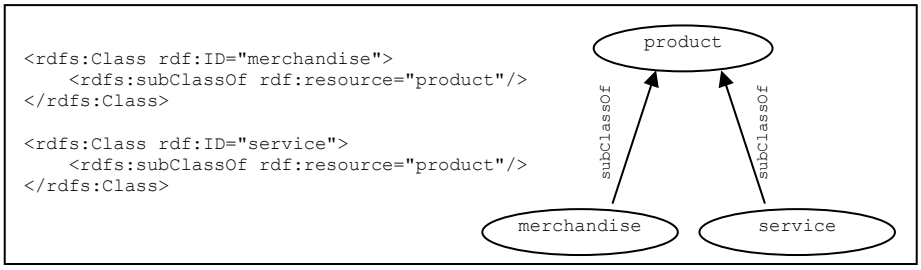

Figure 2. Relation taxonomy

Taxonomies can also be represented as rules of the following very special first-order Horn form for each pair P, P' such that subClassOf(P,P') holds:

```
P'(X) :-P(X).
```

For example subClassOf(P,P') becomes:

```
product(X) :- merchandise(X).
```

Definition 2

Let P and P′ be two relations. P and P′ are 'sibling-interoperable' if they are not on the same path in a relation taxonomy but have a common ancestor relation.

For example:

```
P = merchandise
P′ = service
```

Based on the information in the taxonomy (Figure 2) `merchandise` is `subClassOf` of `product` and `service` is `subClassOf` of `product`; we then conclude that `P` and `P′` are 'sibling-interoperable'. However, one may argue that all the nodes in a taxonomy-tree have a common ancestor node (i.e., root node). Therefore, `P` and `P′` are considered only 'sibling-interoperable' if the *taxonomic similarity* between them does not exceed a threshold. Work on measuring the similarity between terms in a taxonomy has been done in [2, 13].

Homonyms: When the same terms (e.g., relations in rules) refer to different real world objects or concepts, they are known as homonyms. For example, the following rule defines the occurrence of an `item` in a list:

```
item(X) :- list(Y,X).
```

Obviously, `item` here is different from `item` in RB_2. This is called the homonym conflict for `item`. There are several techniques to detect homonym conflicts. First, we can examine the arities of relations that have the same name. Two or more relations `P` with different arities can be resolved into different relations (relations `P/0, P/1, P/2,...,P/n`). For example, `merchandise/1` and `merchandise/2` would be different relations, so there is no homonymity problem for them. Second, we can also further check the datatypes of arguments in homonymous relations. Finally, we can investigate the equivalence of the body relation calls in two rules containing homonymous relation in the head. For example, we can try to determine that `list(Y,X)` and `supplier(Y,X), store(Y,Z)` are not equivalent. The more criteria we use, the more information we have to resolve conflicts between relations accurately.

b. Structural Conflicts

Different rulebase providers formalize an issue in a different number of rules and each rule may have a different structure. For example, a rulebase provider R_3 can declare `item` for the rulebase RB_3 as follows:

RB_3

```
item(X) :- from(Y,X,Z).
from("Compact Corp.", "Printer", "Boston").
```

Here, from in RB_3 is the aggregation of supplier and store in RB_2. In this case, a comparison between RB_3 and RB_1 is much harder than that between RB_2 and RB_1. Besides the *aggregation* conflict, other subtypes of structural conflicts such as *missing-item* and *generalization/specification* found in database integration [7, 8] also occur here. The resolution for these types of conflicts can be adapted from those of database integration approaches [7, 8].

c. Datatype Conflicts

An important criterion for verifying whether two relations match or not is the matching of the datatypes of their arguments. Since relations often go with arguments, the opportunity for interoperability between two relations, namely H and H′, will be higher if there exist some relationships between the datatypes of their arguments (we will use the notation argument:datatype)

For example:

```
H = merchandise(X:Machinery)
H′ = product(X:Tools)
```

Suppose that we are uncertain whether H matches H′ or not. There may exist some metadata represented in terms of RDFS (Figure 3) about the relationship between Machinery and Tools as follows:

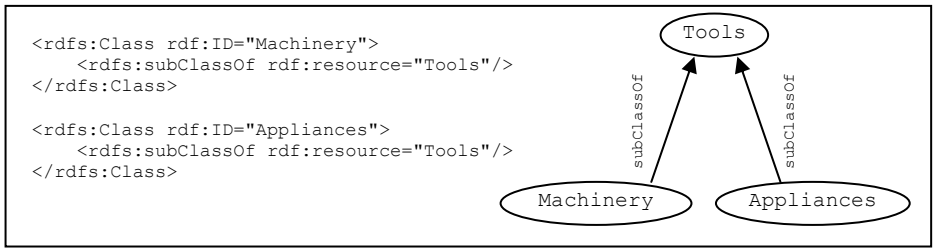

Figure 3. Taxonomy about relationship between objects

which means Machinery is subClassOf of Tools. We can then conclude that H matches H′ since their relation names are *'subsumption- interoperable'* and the datatype of the argument in one relation is related to or matches that of the other (i.e., Machinery vs. Tools).

2.1.2. Integration of Rulebases Having Transformable Expressive Fragments

Different rulebases can interchange their transformable expressive fragments. We classify the integration of rulebases having transformable expressive fragments into two main types, namely *positional vs. slotted logics* and *binary vs. n-ary relations*. In this paper, we only discuss the positional vs. slotted logics of Figure 1.

Positional vs. Slotted Logic

A piece of information can be presented in either a slotted logic language or a positional logic language. For example, a statement:

```
"John pays three hundred US dollars for his meal"
```

is represented in the slotted style of POSL[1] as follows:

$R_{slotted1}$

```
pay(buyer->"John"; item->"meal"; USD-price->300).
```

But when these data are positionalized, they become:

R_{pos1}

```
pay("John", "meal", 300).
```

Obviously, with the slotted representation, data can be described in a more semantic way with additional information carried on by slots. When integrating positional and slotted rulebases, there is a certain risk, by which information can be lost or need to be fulfilled by the system. For example, consider a rule $R_{slotted2}$ as follows:

$R_{slotted2}$

```
pay(buyer->"John"; item->"meal"; CAD-price->300).
```

which means that

```
"John pays three hundred Canadian dollars for his meal".
```

If R_{pos2} is a positionalized version of $R_{slotted2}$, it will become:

R_{pos2}

```
pay("John", "meal", 300).
```

Here, even though $R_{slotted1}$ and $R_{slotted2}$ are different, their respective positionalized versions are the same. This is due to a certain amount of lost information (metadata) when we transform from a slotted logic language to a positional logic language. In the opposite direction, from a positional logic language to a slotted language, additional information must be provided. For example, when transform R_{pos2} to a slotted version, we will not know exactly what should be the appropriate slot name for each data item. Thus, slotted versions of R_{pos2} can be $R_{slotted1}$, $R_{slotted1}$ or others. A possible solution for this problem is that instead of removing slot names when we positionalize rulebases, we could keep them in the form of `Signature` declarations [5, 9] for further use.

---

[1] http://www.ruleml.org/submission/ruleml-shortation.html

2.1.3. Integration of Rulebases Having Different Expressive Fragments

Rulebases of the same language can also be expressed in different fragments of expressiveness. In some family languages, such as RuleML, a rule can be from the lowest fragment of semantics, namely `binarydatagroundfact` (fact in a binary format), to the highest fragment of semantics, namely `naffologeq` (first-order logic with negation of failure and equality). Therefore, when integrating rulebases having different expressive fragments, we need a mechanism to utilize the maximum subset of these two rulebases. Basically, rulebase integration of different expressive fragments can be classified into two main types, namely *Datalog vs. Horn logic* and *FOL (First-Order Logic) vs. HOL (Higher-Order Logic)*. Due to space limitations, only the issue of Datalog vs. Horn logic of Figure 1 is discussed.

Datalog vs. Horn logic

Different users can formulate their rulebases with different level of semantics from the same issue. Returning to the previous example `merchandise`, a rulebase provider R_4 can express his rulebase RB_4 as follows:

RB_4

```
merchandise(X) :- provider(company[Name, Loc], X),
                  warehouse(Name,addr[Street,City]).
provider(company["Compact Copr.", "USA"], "Printer").
warehouse("Compact Copr.", addr["23 Main Street", "Boston"]).
```

Unlike RB_1, RB_2 and RB_3, which are serialized in Datalog, RB_4 is represented as a set of Hornlog rules. In general, the two following situations can occur:

(1) A less expressive rulebase `R` (e.g., in Datalog) is sent to a more expressive rule engine `E` (e.g., built for Hornlog) to be executed. In this case, the engine `E` can naturally handle `R`, although the engine `E` may not be efficient for this special case.

(2) A more expressive rulebase `R` (e.g., in Hornlog) is sent to a less expressive rule engine `E` (e.g., built for Datalog) to be executed. In this case, the engine `E` cannot handle `R`. However, splitting rulebase can be used to at least interchange the maximum subset of rulebases that is not more expressive.

Returning to our example, if RB_1, RB_2 and RB_3 are sent to a Hornlog rule engine, which is able to handle RB_4, they can be executed naturally. However, the opposite direction does not work since a Datalog rule engine, which is able to handle RB_1, RB_2 and RB_3, cannot handle a Horn rule RB_4.

3. Rulebase Integration Framework for Interoperation and Interchange

From the classification in the previous part, we see that distributed rulebases are heterogeneous in terms of different languages and levels of expressiveness. Based on that classification, a framework for rulebase integration, as rulebase interoperation and rulebase interchange, is proposed. Figure 4 shows a framework for rulebase integration. There are several different kinds of rulebases in this framework. We use traditional rule languages such as F-Logic [9], Prolog [16] and Relfun [3] to describe three participating legacy rulebases, and we use XDD [22] and RuleML, two XML syntax-based rule languages, to model two other participating rulebases. These rulebases are distributed and heterogeneous in both languages and levels of expressiveness but their expressiveness are also intersected.

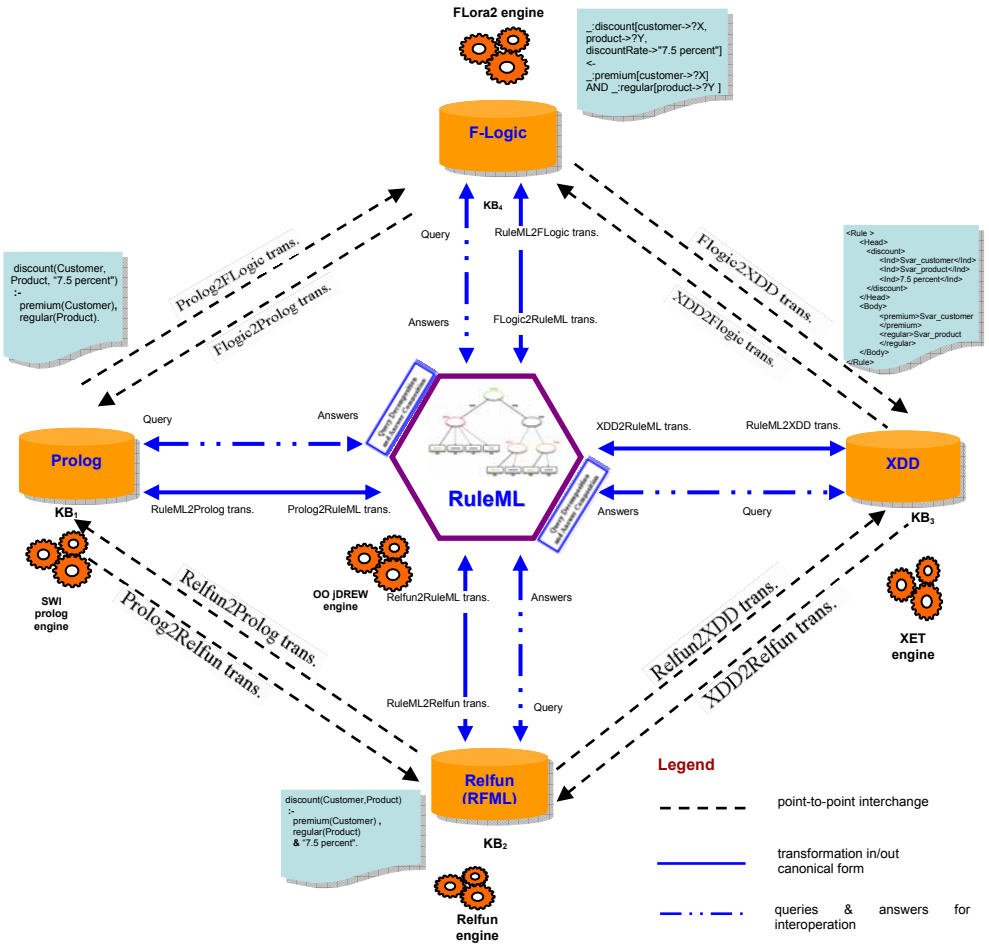

Figure 4. Framework for rulebase integration

In our framework, rulebase integration can be done by either interoperation or interchange. While interoperation (bold dashed-dot-dot line) supports query transformation, distributed querying, and answer composition for distributed (and often autonomous) rulebases, interchange (bold dashed line) transforms heterogeneous rulebases into a canonical form, thus supporting uniform querying. For example, a user's query, in terms of RuleML, asks for data from these existing rulebases. Using the interoperation approach, the system both queries the RuleML rulebase locally and sends this query to the distributed rulebases (e.g., F-Logic rulebase, Prolog rulebase, Relfun rulebase, and XDD rulebase). Data extracted from these distributed rulebases are composed and returned the final results to users. Using the interchange approach, the system first analyzes all the existing rulebases, imports, transforms them into a canonical form and stores them into a central rulebase (e.g., RuleML rulebase). This process is done one and for all. Whenever users pose queries, these queries will be processed locally in the canonical rulebase. With the difference between interoperation and interchange, we find that the classification of rulebase integration (in Section 1) is applicable only for interchange approach, where differences of surface syntaxes and expressive fragments are analyzed.

In the interchange approach, transformation can be done declaratively by using transformation rules, which themselves are interchangeable. This transformation can be total or partial, in that information may be preserved or lost through the transformation. Section 4 will discuss this issue in more detail. In the interoperation approach, an input query is decomposed into subqueries for execution in distributed rulebases. Answers from those local rulebases are then composed into a global one and returned to users. By following the interoperation approach, rulebases can be kept unchanged and executed in an environment best suited for a specific (sub)task but queries have to be processed (repeatedly). By following the interchange approach, entire heterogeneous rulebases have to be transported and transformed into a homogeneous form (once), but this facilitates uniform querying and optimization. The following section will focus on rulebase interchange, but we refer to [19, 20] for details on rulebase interoperation.

4. Issues in Rulebase Interchange

This section will discuss semantics-preserving transformation and serializing functions, two important issues of rulebase interchange as well as give initial experimental rulebase interchange results.

4.1. Semantics-Preserving Transformation

When transforming a rulebase encoded in (the surface syntax of) a language to another language, an important issue is how to preserve the semantics of that rulebase. Ideally, information is preserved during transformation. However, in some situations, we have to accept *lossy transformation*, which means that some information may be lost during a transformation of a rulebase from a rule language to another one. Figure 5 shows a commutative diagram of rulebase transformation corresponding to the XDD -> RuleML

interchange of Figure 4. If `trans` is a transformation function from XDD to RuleML, we want to have `trans` as a homomorphism yielding the following commutative diagram:

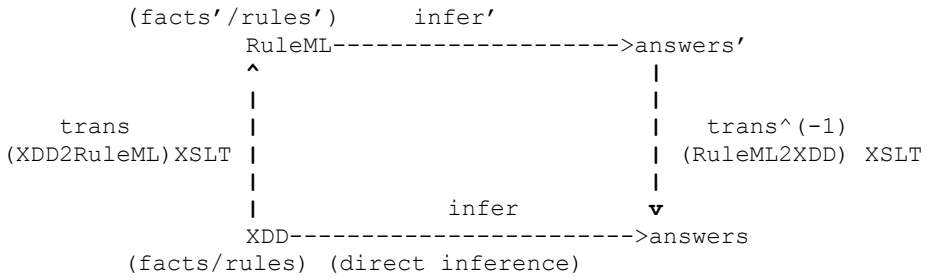

Figure 5. Commutative diagram of rulebase interchange

In this commutative diagram, the answers of the indirect inference and those of the direct inference should be equivalent. This means that `facts/rules` after having sequentially applied `trans, infer'` and the `trans` inverse `trans^(-1)` should produce the same answers as when having applied `infer` directly. The above diagram can thus also be expressed by the following formula:

```
trans^(-1)(infer'(trans(XDD)))= infer(XDD)
```

Errors can appear in either of the two paths: `trans -> infer' -> trans^(-1)` or in `infer`. However, if the source language and the target language (in this example, they are XDD and RuleML respectively) are based on perfect rule engines, then the possibility for errors occurring in `infer` and `infer'` is zero. Thus, `trans` and `trans^(-1)` are the places where errors have occurred.

4.2. Serializing Functions

Most existing languages, including Functional Programming, Logic Programming and Functional Logic Programming Language, employ some versions of functions (constructors, built-in or user-defined functions). Therefore, there arises a need for representing functions while exchanging them on the Web. This can be benefit from Functional RuleML [4], a newly derived sublanguage of the RuleML family language since at the end of 2005. In Functional RuleML, a function can be `interpreted`, `uninterpreted` or can even be `semi-interpreted` for flexibility. For example, if the function of the term `addr(Street,City)` is uninterpreted, the term just denotes the data structure consisting of its *constructor* `addr` applied to its arguments `Street` and `City`. We will emphasize this by using Relfun-like square brackets as in `addr[Street,City]`. The example can thus be marked up as an uninterpreted function (`in="no"`) as follows:

```
<Expr>
   <Fun in="no">addr</Fun>
   <Ind>Street</Ind>
   <Ind>City</Ind>
</Expr>
```

Moreover, since higher-order functions are implemented in many rule-based systems, it is reasonable for Functional RuleML to support them. For example, the following function `CustomerAffiliation` takes the `addr` and `email` functions as its arguments. Since functions (`addr` and `email`) here play the role of arguments of another function `(CustomerAffiliation)`, this is a higher-order function, specifically a higher-order constructor.

```
CustomerAffiliation[addr, email].
```

The markup version of the `CustomerAffiliation[addr, email]` application is as follows:

```
<Expr>
   <Fun in="no">CustomerAffiliation</Fun>
   <Fun in="no">addr</Fun>
   <Fun in="no">email</Fun>
</Expr>
```

By using RuleML consisting Functional RuleML as a canonical form for rulebase integration, we can naturally handle the problem of function representation and interchange.

4.3. Experimental Results

We have developed some XSLT stylesheets to transform between rulebases of RFML and RuleML, RuleML and XDD.

4.3.1. Interchange Between RFML and RuleML

In August of 2005, a `RFML2RuleML.xslt` stylesheet [11] was developed by Jie Li to transform the logical part of RFML (Hornlog RFML) to RuleML 0.89. A usecase, namely Chemical XML Elements (ChemXelem) [11], containing information about all chemical atoms, was used to verify the correctness of the transformation. With the incorporation of Functional Programming into RuleML, a second stylesheet [4] was written to transform the functional part of RFML to RuleML (i.e., Functional RuleML). Similar to the earlier stylesheet, numerous examples were used to verify the correctness of the transformation. Using these two stylesheets, users can transform a whole RFML program consisting of functional and logical parts to RuleML.

4.3.2. Interchange Between XDD and RuleML

RuleML is a very powerful logical/functional language which can describe Datalog, Horn logic, first-order logic as well as higher-order logic. Because of its popularity, RuleML has become a candidate for a standard rule representation on the web. However, it still lacks the capability of modeling user-defined XML documents. On the contrary, XDD has an expressive power on modeling XML documents but it cannot compare to RuleML about the mathematical and computational power. Therefore, by interchanging rulebases of these two languages, these rulebases can first be processed in one environment before passing to the other to process. We can thus exploit the power of both RuleML and XDD. However, since the expressiveness of the two languages is not the same, therefore the transformation here is the only `partial` transformation. Two stylesheets to transform from XDD to RuleML and vice versa were implemented in `<http://www.ruleml.org/usecases/XDD/>`.

5. Conclusions

Rulebase integration has recently gained a lot of attention since it is a basic problem underlying many Semantic Web applications, such as in eCollaboration, enterprise information integration, and semantic query processing. In this paper, we defined the objects and vocabularies of eCollaboration by rules (including taxonomies). We propose a classification of rulebase integration discussing the conflicts of each classification in detail. Several earlier techniques for database integration can be applied to rulebase integration since the former can be regarded as a special case of the latter. From the classification, we presented a unified framework for rulebase integration containing both the interoperation and interchange approaches. Using our framework, a rulebase can be interoperated unchanged (to accommodate legacy rulebases and permit decentralized inferencing) or interchanged via a canonical form before possibly merging them with other ones (to simplify future processing and permit uniform inferencing). We discussed the interchange homomorphisms for preserving the semantics of rulebases on transformation. The integration of various function types in rulebases is enabled by Functional RuleML. Finally, XSLT stylesheets to transform between RFML and RuleML, between XDD and RuleML led to initial interchange results.

References

[1] AnHai, D., and Halevy, A. Y. (2005). Semantic integration in the Database Community: A Brief Survey. AI Magazine, Special Issue on Semantic Integration.

[2] Bhavsar, V.C., Boley, H., and Lu, Y. (2004). A Weighted-Tree Similarity Algorithm for Multi-Agent Systems in e-Business Environments. Computational Intelligence, 20(4): 584-602.

[3] Boley, H. (1999). A Tight, Practical Integration of Relations and Functions. Springer.

[4] Boley, H., et al . (2005). Functional RuleML. http://www.ruleml.org/fun/rfml2ruleml.xslt.

[5] Boley, H., Tabet, S., and Wagner G. (2001). Design Rationale for RuleML: A Markup Language for Semantic Web Rules. Proc. of SWWS'01, The first Semantic Web Working Symposium, Stanford University, California, USA, 381-401.

[6] Bugliesi. M., Lamma. E., Mello. P.(1993). Modularity In Logic Programming. Journal of Logic Programming. Vol. 19-20: 443-502.

[7] Duong, D.D., and Thuy, L.T.T. (2005). Classification and Reconcilement of Conflicts between Heterogeneous XML Schemas. Proc. of the 10th Conference on Artificial Intelligence and Applications, TAAI.

[8] Duong, D.D., and Wuwongse, V. (2003). XML Database Schema Integration Using XDD. Proc. of Advances in Web-Age Information Management Conference, China: Lecture Notes in Computer Science, Springer Verlag, Vol. 2762: 92-103.

[9] Kifer, M., Lausen, G., and Wu, J. (1995). Logical Foundations of Object-Oriented and Frame-Based Languages. Journal of ACM, 741 – 843.

[10] Krovvidy, S., and Bhogaraju, P. (2005). Interoperability and Rule Languages. W3C Workshop on Rule Languages for Interoperability. Washington, D.C., USA. http://www.w3.org/2004/12/rules-ws/paper/54/.

[11] Li, J., et al. (2005). RuleML Use Case ChemXelem. The Periodic System of the Elements. http://www.ruleml.org/usecases/chemxelem.

[12] Li. J., Boley H., Bhavsar, V.C., Mei. J. (2006). Expert Finding for eCollaboration Using FOAF with RuleML Rules. Proc. of the 2006 Montreal conference on eTechnologies, 53-65.

[13] Lu, Y., Ball, M., Bhavsar, V.C., and Boley, H. (2006). Weighted Partonomy-Taxonomy Trees with Local Similarity Measures for Semantic Buyer-Seller Match-Making". Journal of Business and Technology (to appear).

[14] O'Connor, M., Knublauch, H., Tu, S., Grosof, B. N., Dean, M., Grosso, W., and Musen, M. (2005). Supporting rule system interoperability on the semantic web with SWRL. Proc. 4th International Semantic Web Conference, Galway, Ireland, 974-986.

[15] RIF's Design Goal Categories http://www.w3.org/2005/rules/wg/wiki/UCR/Design_Goals.

[16] SWI-Prolog. http://www.swi-prolog.org/.

[17] The Rule Interchange Format (RIF) Working Group http://www.w3.org/2005/rules/Overview.html.

[18] Thuy, L.T.T., and Duong, D.D. (2004). Integration of XML Databases, The Journal of Hue University – Vietnam, Vol. 22.

[19] Thuy, L.T.T., and Duong, D.D. (2005). Query Decomposition Using the XML Declarative Description Language. Proc. of International Conference of Computational Science and Its Applications, Singapore: Lecture Notes in Computer Science, Springer Verlag, Vol. 3481: 1066-1075.

[20] Thuy, L.T.T., and Wuwongse, V. (2003). Query Processing of Integrated XML Databases. Proc. of the 5th International Conference on Information Integration and Webbased Applications & Services, Jakarta, Indonesia, 335-344.

[21] W3C Workshop on Rule Languages for Interoperability. (2005) Washington, D.C., USA. http://www.w3.org/2004/12/rules-ws/.

[22] Wuwongse, V., Anutariya, C., Akama, K., and Nantajeewarawat, E. (2001). XML Declarative Description (XDD): A Language for the Semantic Web. IEEE Intelligent Systems, Vol. 16(3): 54-65.