ORIENT: Integrate Ontology Engineering into Industry Tooling Environment*

Lei Zhang[1], Yong Yu[1], Jing Lu[1], ChenXi Lin[1],
KeWei Tu[1], MingChuan Guo[1], Zhuo Zhang[1],
GuoTong Xie[2], Zhong Su[2], and Yue Pan[2]

[1] APEX Data and Knowledge Management Lab,
Department of Computer Science and Engineering,
Shanghai JiaoTong University, Shanghai, 200030, China.
{zhanglei,yyu,robertlu,linchenxi,tkw,gmc,zhuoz}@apex.sjtu.edu.cn

[2] IBM China Research Lab
NO.7, 5th Ave., ShangDi, Beijing, 100085, China.
{xieguot,suzhong,panyue}@cn.ibm.com

Abstract. ORIENT is a project to develop an ontology engineering tool that integrates into existing industry tooling environments – the Eclipse platform and the WebSphere Studio developing tools family. This paper describes how two important issues are addressed during the project, namely tool integration and scalability. We show how ORIENT morphs into the Eclipse platform and achieves UI and data level integration with the Eclipse platform and other modelling tools. We also describe how we implemented a scalable RDF(S) storage, query, manipulation and inference mechanism on top of a relational database. In particular, we report the empirical performance of our RDF(S) closure inference algorithm on a DB2 database.

1 Introduction

Ontology is the backbone that provids semantics to the Semantic Web. The development of the Semantic Web thus depends heavily on the engineering of high quality ontologies. Numerous tools supporting this task have been developed in the past such as Protégé [1], OilEd [2] and WebODE [3]. Based on the R&D experiences and results of these tools, we initiated the ORIENT (Ontology engineeRIng ENvrionmenT) project[3] that develops an ontology engineering tool that integrates into existing industry tooling environments – the open-source Eclipse platform and the WebSphere Studio™ developing tools family.

In this paper, we describe how two important issues are addressed during the project, namely tool integration and scalability. The importance of tool integration comes from three requirements on the tool and can not be underestimated.

* Opinions expressed and claims made in this paper are of the authors. They do not represent the opinions and claims of the International Business Machine Corporation.

[3] http://apex.sjtu.edu.cn/projects/orient/, also available from IBM Semantics Toolkit at http://www.alphaworks.ibm.com/tech/semanticstk

First, to be a complete ontology engineering tool, it must integrate different tools for different sub-tasks of ontology engineering such as building, mapping, merging and evaluation. Second, because applying ontologies in real applications requires the tool be used together with other tools to accomplish a certain task, tool integration is thus required to reduce the cost of switching between different tools. Finally, if the tool can be integrated with exisiting tooling environment e.g. the Eclipse platform and the WebSphere Developing tools family, it will be much more developer-friendly and easier to use. The success story of the Protégé tool on the integration of numerous plugins also remind us on the advantages of tool integration. In the ORIENT project, instead of designing the tool as a stand-alone application, we decided to build it as a set of Eclipse plugins that make it morph into the Eclipse platform. This strategy proves to be effective in satisfying the above requirements and achieves both data and UI level integration with other tools. In section 5, an example showing how ORIENT can be integrated with an UML editing tool is presented. Futhermore, because WebSphere Studio developing tools are also built on the Eclipse platform, this strategy paves a direct way to the product family.

In addition to tool integration, scalability is another important issue in ORIENT. Since ontologies[4] in real applications tend to have a large size, managing large volume ontology in industry applications is thus not a feature but a necessity. In this paper, we describe how we implemented a scalable RDF(S) storage, query, manipulation and inference mechanizm on top of a relational database and report the empirical performance of this implementation. In particular, we describe how an algorithm is carefully designed to calculate RDF(S) closure on a relational database and give the experiment result of its performance test. To our best knowledge, we do not know any previous report on RDF(S) closure inference performance on databases yet, although there are reports on RDF(S) storage and query performance on databases e.g. RQL[4] and Jena2[5].

The rest of the paper is organized as follows. Section 2 introduces the Eclipse platform and section 3 describes the Eclipse-based ORIENT architecture. We then describe the integration with Eclipse UI in section 4 and the integration with other modelling tools in section 5. The RDF(S) storage, query, manipulation and inference implementation and empirical performance analysis is presented in section 6. We conclude this paper in section 7.

2 Eclipse Tools Integration Platform

As stated in its technical overview [6], Eclipse (http://www.eclipse.org) is an open extensible IDE for anything and nothing in particular. It provides a unified mechanism to expose extension points and develop plugins that leads to seamlessly-integrated tools. The Eclipse architecture is designed around the concepts of plugins, extension points and extensions. New tools and functionalities

[4] In this paper, the term *ontology* refers to a knowledge base that includes concepts, relations, instances and instance relations that together model a domain

can be integrated into the platform as plugins. Through this plugin architecture, Eclipse platform thus provides a solid foundation for tools integration.

Besides its well-designed plugin architecture, the Eclipse platform also offers an extensible user interface paradigm for various plugins to share and extend. The UI paradigm is based on editors, views and perspectives. A perspective groups and arranges a set of editors and views on the screen. New types of editors, views and perspectives can be added to the platform UI through its plugin mechanism. Actually, ORIENT adds its own perspective with 8 views and editors to the Eclipse platform.

The above gives a very brief description of the Eclipse platform. We refer interested readers to the technical overview [6] and various articles on the Eclipse.org web site for more in-depth information about the Eclipse platform. The WebSphere Studio developing tools are also built on the Eclipse platform.

3 ORIENT Architecture

In the design of the ORIENT architecture, the most influential factor is tool integration at both the data and UI level . The demanding need for tool integration actually arises from three aspects. Firstly, ontology engineering has a complex life cycle [7, 8] that needs different tools for different sub-tasks (e.g. building, mapping, merging and evaluation, etc.). Although the sub-task tools may be developed by different people at different time, they must be integrated to form a complete ontology engineering environment. For example, it will be preferable if the ontology evaluation tool can be tightly integrated with the ontology building tool to provide real-time feedback on the quality of the ontology.

Secondly, applying ontology in real applications requires that the ontology engineering tool be used together with other tools to accomplish a certain task. For example, annotating Web pages with semantic tags needs both an ontology browsing tool and an annotation tool. Building and testing a Semantic Portal [9] may involve ontology building, portal customization, and portal generation tools (e.g. in OntoWeaver [10]). Users of these tools have to switch back and forth between them to get their work done. If the tools can be seamlessly integrated at both the data and UI level in a single development environment, the cost associated with switching between them will be greatly reduced. In addition, if other modelling tools (e.g. ER and UML tools) can be integrated, it will enable users not familiar with ontology engineering to use our tool through another familiar tool. It can also leverage existing knowledge on data and application modelling to help them learn and get familiar with ontology engineering. This actually is in accordance with many researchers' opinion that the Semantic Web should come as an evolution rather than a revolution.

Finally, it will be much more developer-friendly and easier to use, if the tool can be integrated with the Eclipse platform or the WebSphere Developing tools product family and appear not as a brand new tool for users to learn but as a new integrated set of functionalities. In addition, the user base of the industry tooling environments will also help the ontology engineering tool reach a wider

developer community. We demonstrate how ORIENT morphs into the Eclipse platform in section 4.

Based on the above considerations, we decided not to design the tool as a new stand-alone application but as a set of loosely-coupled cooperative Eclipse plugins. The plugins are organized into a three layer architecture as shown in Fig.1. RDF related plugins are used in Fig.1 as concrete examples. The archi-

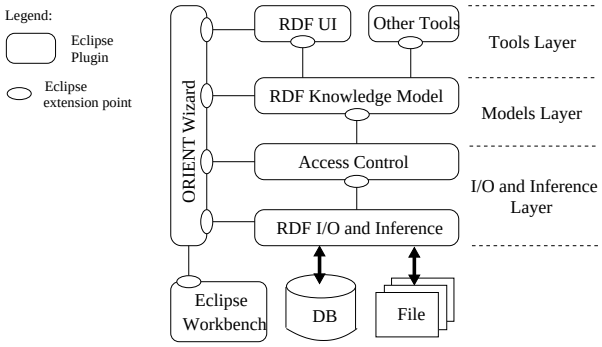

Fig. 1. The ORIENT architecture using RDF plugins as an example

tecture layers are defined and separated by Eclipse extension points. Plugins in each layer are only required to correctly extend (implement) the extension points. This enables flexible mix-and-match of different plugins (possibly from different vendors) to compose an ontology engineering tool. For example, one may choose to use an RDF DB plugin to provide storage function for an OWL knowledge model plugin. Actually, the ORIENT Wizard plugin (at the left side of Fig.1) does exactly the job of helping users compose and manage plugins from different layers.

In the architecture, the models layer provides an in-memory model and access API for the ontology and it enables the sharing of the same model among different tools in the tools layer. For tools using different models, a translation between models is needed to integrate them at the data level. An example of this is shown in section 5.

Note that the lowest layer in the architecture is I/O and inference layer that manages the storage and inference of ontologies. Because it is usually hard to load a very large ontology entirely into the main memory, better inference algorithm can be designed if we know how the ontology is stored on the permanent storage. At the same time, the design of the inference algorithm may also impose certain requirements on the storage schema. Therefore, we purposely put the inference function down in this layer for better scalability.

Finally, this Eclipse plugin-centered architecture makes ORIENT seamlessly morph into the Eclipse platform. This greatly improves the tool's usability and lays a foundation for tool integration.

4 Integration with the Eclipse Platform

In this section, we show several examples of the UI level integration of the ORIENT tool with the Eclipse platform.

First, ORIENT uses an ontology file to represent an ontology in the Eclipse platform. The file stores information such as which plugins should be activated to access the ontology, where the ontology is stored, etc. Once the file is opened, an ORIENT perspective with editors and views will be opened for editing the ontology. In this way, ontologies can be put as files into any existing Eclipse project and sit side-by-side with other resources in the project. This makes incorporating, using and managing ontologies in a project very easy. Furthermore, switching between ORIENT and other tools can be done by simply double-clicking on different resources in an Eclipse project. This is where tool integration helps user quickly switch between different tools to accomplish a task in a project. In addition to ontology files, ORIENT also supports creating ontology projects that contain one or more ontologies. Fig.2 shows the ORIENT New Ontology File/Project Wizard in the WebSphere Studio Application Developer V5.1.

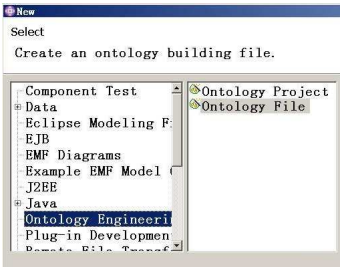

Fig. 2. New Ontology File/Project wizard

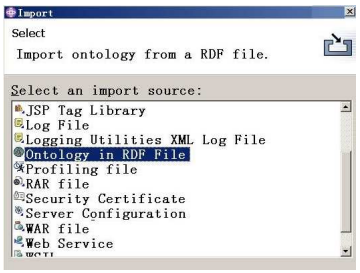

Fig. 3. RDF Import/Export Wizard

Second, ORIENT provides an ontology import/export function for users to import/export ontologyes stored in ORIENT from/to external files. This import/export functionality is also integrated with the general Eclipse import/export UI and can be accessed very easily. Fig.3 shows a screenshot of the ORIENT RDF import function entry in the WebSphere Studio Application Developer V5.1 import UI.

Finally, ORIENT offers a dedicated Eclipse perspective that groups its ontology related views and editors. Fig.4 is a screen shot of the ORIENT perspective. Currently, the ORIENT perspective contains one editor and eight views for users to view and edit an RDF ontology. The eight views are grouped into list views, hierarchy views and information views. The three list views (i.e. *Resource List*, *Class List* and *Property List*) show all the existing resources, classes and properties respectively. The two hierarchy views (i.e. *Class Hierarchy* and *Property Hierarchy*) display the hierarchy trees of classes and properties. The three information views are used to display detailed information of a resource, a class or a

property respectively. The EclipseUML editor in the top right part of the figure is explained in the next subsection.

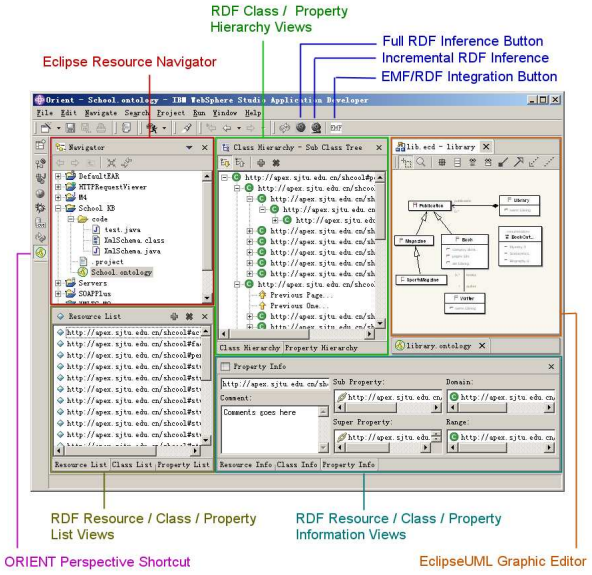

Fig. 4. The ORIENT Perspective with EclipseUML

5 Integration with Other Tools

The above section has demonstrated the UI level integration of ORIENT with the Eclipse platform. What may be more important is how ORIENT is integrated at the data level with other tools. In this section, we show how ORIENT can be integrated at the data level with Eclipse tools that are based on the Ecore model.

Ecore model is a central part of the Eclipse Modeling Framework (EMF)[5]. Currently, it provides the class digram modeling capabilities of UML. For example, Ecore provides modeling elements such as `EClass, EAttribute` and `EReference` etc. Several tools have been developed (or are under development) based on the Ecore model. Eclipse EMF plugin is a tool that provides a Java framework and code generation facility for building tools and other applications based on the Ecore model. EclipseUML[6] is a simple graphical UML editing tool

[5] Ecore and EMF can be found at http://www.eclipse.org/emf
[6] EclipseUML can be found at http://www.omondo.com

that bases its model on Ecore. UML2 is an Eclipse tool project[7] that develops an Ecore and EMF-based tool for the UML 2.0 meta-model.

As we have mentioned in section 3, if modeling tools such as UML can be integrated with ORIENT, it will enable users not familiar with ontology engineering to use ORIENT through another familiar tool (e.g. UML tool) and it can also leverage existing industry knowledge on data and application modeling to help users learn and get familiar with ontology engineering. This is the main motivation to integrate ORIENT and the Ecore model based tools. The actual data level integration is done through a model translator. The model translator works between the ORIENT model layer and the Ecore model. It captures the two sides' model changes, translates the changes and applies them to the models accordingly so as to keep them synchronized.

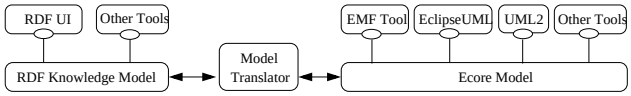

Fig. 5. The Integration of ORIENT and Ecore Tools

Fig.5 shows this idea visually. This integration enables users to utilize different tools to work on the same model even if the model is expressed in different formalisms. Users can, for example, first edit the model in a graphical UML tool (e.g. EclipseUML) and then switch to ORIENT to view it through the RDF perspective and finally export it in RDF. On the contrary, users may also be able to visualize an RDF model through a graphical UML tool and save it as a UML model. This demonstrates the power of tool integration and how it benefits the users.

However, the difficulties come from maximally preserving the model semantics in the model translator. Currently, we are doing a translation between RDF and Ecore in ORIENT. Detailed explaination of this translation is beyond the scope of this paper. The following lists some important translation rules in the current implementation:

- `EClass` elements of Ecore are converted as `rdfs:Class` resources and their hierarchical relationships are preserved using `rdfs:subClassOf`.
- `EAttribute` elements in Ecore are converted as `rdf:Property` resources.
- EMF elements of types other than `EClass` and `EAttribute`, when converted as RDF resources, take on their Ecore types(such as `EReferences`) as the value of their `rdf:type` properties.

The current translation is by no means perfectly sound and complete. It is a further research issue in ontology transformation and mapping and in ontology development using UML [11].

[7] UML2 can be found at http://www.eclipse.org/uml2

Aided by the UI level tool integration, the results of the data level tool integration are immediately visible to the end users. In Fig.4, ORIENT is actually used together with EclipseUML. Both the ORIENT views and the EclipseUML graphical editor are shown to the user (note the EclipseUML graphical editor at the top right corner). Changes in each tool's view will be instantly reflected in the other tool's view once the they are saved.

6 RDF(S) Storage, Query, Manipulation and Inference

6.1 Query and Manipulation through RQML

In the current ORIENT I/O and Inference Layer, a Default ORIENT RDF DB plugin is implemented to provide RDF(S) storage, query, manipulation and inference capabilities to the upper layers. For both users and programs to query and manipulate RDF data in an uniform manner, a declarative RDF Query and Manipulation Language (RQML) is defined in the plugin. Based on RQML, RDF models on top of the I/O and inference layer can provide higher level object-oriented APIs for easier program access. RQML is designed based on previous RDF query languages such as RQL[4], RDQL[5] and SeRQL[12].

RQML tries to be maximally syntax-compatible with RDQL while at the same time borrows features like path expression from SeRQL. In addition, it also introduces some new commands not available in previous languages like INSERT, DELETE and UPDATE. For example,

```
UPDATE (?p, <!http://employee/level>, ‘‘Senior Engineer’’)
WHERE  (?p, <!http://employee/level>, ‘‘Engineer’’),
       (?p, <!http://employee/hasCert>, ‘‘Linux’’).
```

6.2 Inference Capabilities

Reasoning on existing knowledge to discover implicit information is an important process on the Semantic Web. It is also very useful in an ontology engineering tool. Common queries such as what are the (direct and indirect) sub-classes and instances of an exiting class, what instances has an interested relationship with a given instance may all involve inference. According to the RDF Semantics document [13], RDF(S) closure contains all the triples that can be inferred from an RDF(S) ontology. In ORIENT, an algorithm for calculating the RDF(S) closure is implemented to provide RDF(S) inference support. As we have metioned in section 3, the inference algorithm is purposely embedded in the Default ORIENT RDF DB plugin. This improves the performance of the inference algorithm on a relational database. In section 6.5, we report the algorithm's empirical performance on a DB2 database. To our best knowledge, we do not know any previous report on RDF(S) closure inference performance on databases yet, although there are reports on RDF(S) storage and query performance on databases e.g. RQL[4] and Jena2[5].

The RDF(S) closure calculation is based on a core subset of the RDF(S) entailment rules defined in the RDF Semantics document [13] and can be used

in either full mode or incremental mode. In the full mode, RDF(S) closure is not calculated until the plugin is told to do so and a full re-calculation is performed. In the incremental mode, every modification to the RDF(S) ontology will trigger a calculation on the change to the RDF(S) closure. The full mode is suitable for batch update to the ontology and may take a long time to complete while the incremental mode is suitable for fast response to small update. Once the RDF(S) closure is calculated, it can be transparently queried and manipulated through RQML. At the same time, users of RQML can also distinguish between inferred and non-inferred data in the closure and process them differently.

6.3 Storage Schema

In order to support efficient RQML processing and RDF(S) closure calculation, we carefully designed a relational database schema to store RDF(S) ontology. We followed the following principles in the design of the storage schema:

- For high performance RQML processing and RDF(S) closure calculation, we trade storage space for speed.
- Although the storage schema should be able to handle arbitrary RDF(S) data, it must optimize for normal data distribution and typical queries in ontology engineering scenarios.

Table 1. Major database tables

Table Name	**Table Description**
`RDFSubPropSubProp`	This table holds all triples `(x, rdfs:subPropertyOf, rdfs:subPropertyOf)`.
`RDFSubProp`	This table holds all triples `(x, rdfs:subPropertyOf, y)`.
`RDFDomain`	This table holds all triples `(x, rdfs:domain, y)`.
`RDFRange`	This tables holds all triples `(x, rdfs:range, y)`.
`RDFSubClass`	This table holds all triples `(x, rdfs:subClassOf, y)`.
`RDFType`	This table holds all triples `(x, rdf:type, y)`.
`RDFStatement`	This table holds all triples `(x, y, z)`.
`RDFResource`	This table holds all RDF resources.
`RDFUserProp`	This table holds the names of user specified properties.
`RDFProp*`	These tables hold user specified property groups.

Table.1 lists the major database tables in the current storage schema design. Since the `RDFStatement` table already holds all RDF triples, all other tables can be seen as redundant. In addition, we store inferred data with non-inferred data together in the tables. This can also be seen as a redundancy. However, this redundancy design greatly facilitates and speeds up query processing and RDF(S) closure calculation. For example, because inferred data are directly stored, answering queries involving inference is almost as fast as queries without inference.

The drawback of this design is that much care must be taken to maintain consistency among the tables.

To quickly answer typical queries about class hierarchies, property hierarchies/domains/ranges and instance types, the tables `RDFSubClass`, `RDFSubProp`, `RDFDomain`, `RDFRange` and `RDFType` are created. They can quickly provide answers to these typical queries. In order to support RQML literal comparison and calculation, the `RDFLiteralInteger`, `RDFLiteralFloat`, `RDFLiteralBoolean`, and `RDFLiteralString` talbes are created (not listed in Table.1) to hold common data type literals and leverage native SQL data type comparison and calculation. We also borrowed Jena2's Property Tables design [5] in the `RDFUserProp` and `RDFProp*` tables to speed up property lookup.

In addition to the above considerations, the design of the tables are also influenced by and closely related to the RDF(S) closure calculation algorithm which is detailed in the next subsection.

6.4 RDF(S) Closure Calculation Algorithm

The closure calculation algorithm works in either full mode or incremental mode. For brevity, we only describe the full mode here. The algorithm supports a core subset of the RDF(S) entailment rules including `rdf1,rdfs2 -- rdfs11` (as defined in the RDF Semantics document [13]). We call these eleven rules *the rule set* and the involved five RDFS vocabularies `rdfs:domain, rdfs:range, rdfs:type, rdfs:subPropertyOf,` and `rdfs:subClassOf` *the reserved vocabularies*. The rules are selected based on their importance and wide usage in common RDF ontology engineering scenarios. Further development of ORIENT may support more RDF(S) entailment rules.

A well known fast forward-chaining rule algorithm is Rete [14]. It minimizes the impact to the rule system caused by the adding or removing of facts. However, it still falls into repetitive loops if the rules contain cycles. Because the RDF(S) rule set contains cycles, we try to manually remove them as much as possible to obtain an almost one-pass algorithm. By analyzing the rule set, we found that the cycles stem from the following two problems:

P1: The two transitive predicates `rdfs:subPropertyOf` and `rdfs:subClassOf` give rise to cyclic repetitive calculation.
P2: The rules in the rule set and the `rdfs:subPropertyOf` predicate create interdependent derivation relationships among the predicates.

However, scrutinizing the second problem P2 reveals that under certain circumstances the cyclic repetitive calculation can be reduced to just one pass. Fig.6 shows a simplified graph of the predicates' derivation relationships caused by the rule set. Edge lables are names of the rules that cause the derivation relation. Two set of derivation relations are omitted from the graph:

R1: Rules `rdf1`, `rdfs2`, `rdfs3`, `rdfs4a` and `rdfs4b` may derive `rdf:type` from any predicate.

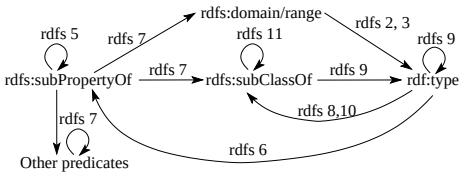

Fig. 6. Derivation Relationships among Predicates in The Rule Set

R2: Rule `rdfs7` may derive any predicate from any predicate due to possible `rdfs:subPropertyOf` relations.

The self loops caused by `rdfs5`, `rdfs11`, and `rdfs9` in the figure actually represent the problem P1. Without considering P1, it is now clear from Fig.6 that `rdfs6, rdfs8, rdfs10` and R2 create big and complex loops in the derivation graph. We now introduce the Original Semantics Assumption that removes these loops.

Original Semantics Assumption Let $\mathcal{A}$ be the set of RDFS axiomatic triples defined in the RDF Semantics document [13]. Let $\mathcal{T}$ be the closure of $\mathcal{A}$ under the rule set. Let Ω be the closure of the current RDF knowledge base under the rule set and $\mathcal{R} \subseteq \Omega$ be the set of all the triples whose subject is in the set {`rdfs:domain, rdfs:range, rdfs:type, rdfs:subPropertyOf, rdfs:subClassOf`}. The original semantics assumption supposes that $\mathcal{R} \subseteq \mathcal{T}$.

This assumption actually assumes that the RDF knowledge base does not strengthen the semantics of the five reserved RDF(S) vocabularies. They still keep their original meaning defined by the axiomatic facts. In most RDF(S) ontology engineering scenarios, this assumption is natural and can be satisfied because most applications does not require the change of the original RDF(S) semantics.

Under this assumption[8], the `rdfs6` and `rdfs10` rules

```
rdfs6:  (uuu rdf:type rdf:Property) → (uuu rdfs:subPropertyOf uuu)
rdfs10: (uuu rdf:type rdfs:Class) → (uuu rdfs:subClassOf uuu)
```

become trivial. If the `rdf:type` closure is obtained, the two rules can be applied only once on the `rdf:type` triples and we can be sure that no more triples can be obtained from them. That is, they can not create loops in the derivation graph. Now let's look at the `rdfs8` rule:

```
rdfs8: (uuu rdf:type rdfs:Class) → (uuu rdfs:subClassOf rdfs:Resource)
```

Under the original semantics assumption, `rdfs:subClassOf` can not be other property's sub-property and `rdfs:Resource` can not be other resource's subclass. Therefore if the `rdf:type` closure is obtained, this rule can be applied

[8] The Original Semantics Assumption is actually stronger than what we really needed. However, it is easier to explain and understand.

only once on the `rdf:type` triples and we can be sure that no more triples can be derived from it. Hence this rule can not create loops in the derivation graph either. We can now safely remove the back lines caused by `rdfs6, rdfs8,` and `rdfs10` from Fig.6.

Finally, let's review R2. Because of the original semantics assumption, the `rdfs:subPropertyOf` relation can only hold from other predicates to the five reserved vocabularies. Hence the `rdfs7` rule can only create derivation relation from "other predicates" to the five reserved vocabularies in Fig.6. If R2 is now drawn on Fig.6, the only loop it can create is the one between the "other predicates" and the `rdfs:subPropertyOf`. However, this loop can be broken because the `rdfs:subPropertyOf` closure actually can be independently calculated[9].

Now, the only loops left in Fig.6 are the self loops. Except for these self loops, our algorithm can thus calculate the entire closure in one pass. The algorithm first adds the RDFS axiomatic facts into the database. It then calculates the `rdfs:subPropertyOf` and `rdfs:subClassOf` closure using a variant of the Floyd algorithm. After that, the closures of `rdfs:domain`, `rdfs:range` and `rdf:type` are obtained in this order without repetitive calculation because all the closures they depend on are already obtained. Finally, we can calculate the `rdfs7` rule closure of the "other predicates". Note that this closure can also be gained without repetitive calculation if we follow the topology order created by the `rdfs:subPropertyOf` relation among the "other predicates". This one-pass algorithm's performance is confirmed by real data experiments shown in the next subsection. The correctness is also verified using W3C RDF test cases.

6.5 Query and Inference Performance

In this section we report the results of the experiments performed to empirically evaluate the performance of query and inference of the current ORIENT RDF DB plugin. The inference performance test is first presented followed by the query performance test.

Two data sets are used in the experiments. One is an artificial data set called "T57" that consists of only `rdfs:subClassOf` and `rdfs:subPropertyOf` relation triples that construct a class hierarchy tree and a property hierarchy tree. Both the two trees have a maximum height of 7 and a constant fan-out of 5. Another data set is "WN", which is the RDF representation of WordNet[10]. All experiments are performed on a PC with one Pentium-4 2.4GHz CPU and 1GB memory running Windows XP Pro, using J2SDK 1.4.1 and Eclipse-SDK-2.1.1 connecting to a local machine DB2 UDB V8.1 Workgroup Server. All data are stored in the database. The inference time is measured as the time cost to transform a database state of containing only original triples to one that also containing all inferred triples. The query time is measured from the time when the query is sent to the time when all results are fetched one by one from the database.

[9] For brevity, the detailed method is not shown here

[10] Available at http://www.semanticweb.org/library/

In our one-pass RDF(S) closure inference algorithm, the major component that determines the order of magnitude of time complexity is the calculation of the transitive closure of `rdfs:subPropertyOf` and `rdfs:subClassOf` using Floyd algorithm on databases. The T57 data set is specifically designed to measure the empirical time complexity of it. By growing the two trees in T57 via adding one level of height each time (with constant fan-out of 5) until the height of 7 is reached, a series of growing data set for inference is got. The result is shown in Fig.7. The T57-1 line shows the relation between the inference time and the number of original triples. The T57-2 line shows the relation between the inference time and the number of triples after inference.

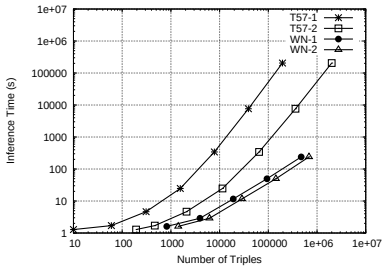

Fig. 7. RDF(S) Inference Performance

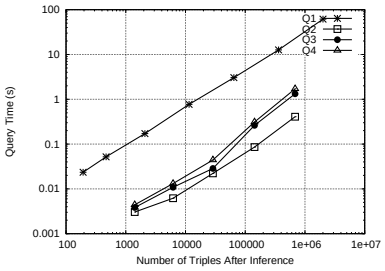

Fig. 8. RQML Query Performance

Different from T57, the WordNet data set has a very small RDF Schema with a very large set of instances and instance relations. The instance data in the four WordNet RDF files are sampled at the same speed. The number of sampled triples are multiplied by 5 each time and the triples from the four files are put together to get a series of growing data set for inference. The result is also shown in Fig.7. The WN-1 line shows the relation between the inference time and the original number of triples and the WN-2 line shows the relation between the inference time and the number of triples after inference.

Both the X-axis and Y-axis of Fig.7 are in log10 scale. The two T57 lines show a linear trend when the number of triples are large (> 1000). Linear regression analysis of the last four points at the end of each T57 line shows that the slopes are 1.87 and 1.75 for T57-1 and T57-2 respectively. This indicates approximately $O(n^{1.87})$ and $O(n^{1.75})$ time complexity. In theory, calculating transitive closure using Floyd algorithm has worst-case time complexity $O(n^3)$. When the algorithm is performed on a database, many factors of the RDBMS may further affect the performance. Combining the experiment result, we tend to empirically predicate that, when performed on a largely tree hierarchy ontology like T57 on a relational database, the time complexity of our inference algorithm is around $O(n^2)$.

Because the WordNet RDF data consists mostly of instance data, its number of inferred triples and inference time are much lower than T57. It, however, shows the same trend of linear relation in Fig.7. Linear regression analysis of the last

four data points at the end of each WN line indicates approximately $O(n^{0.92})$ and $O(n^{0.93})$ time complexity. Similarly, we empirically expect that, when performed on a largely instance data ontology like WordNet on a relational database, the time complexity of our inference algorithm is around $O(n)$.

We can see in Fig.7 that, in the experiments, the number of the triples after inference is in linear proportion to that of the original triples.If a normal RDF(S) ontology satisfies this property and the Original Semantics Assumption in section 6.4, with characteristics between T57 and WordNet, we empirically estimate that its inference time complexity on a relational database is likely between $O(n)$ and $O(n^2)$. The n here can represent either the number of the original triples or the number of the triples after inference.

The RQML query performance is tested on both T57 and WordNet data set with inferred triples. We used the following four queries to test sub-class query, simple query, query with join and query involving literals:

```
Q1: SELECT ?X WHERE (?X, <rdfs:subClassOf>, [aClass])
Q2: SELECT ?X WHERE (?X, <wn:similarTo>, [randomAdjective])
Q3: SELECT ?Y WHERE ([randomNoun] <wn:hyponymOf>, ?X),
                    (?X, <wn:wordForm>, ?Y)
Q4: SELECT ?X WHERE (?X, <wn:wordForm>, ?Y) SUCHTHAT ?Y=[randomWordForm]
```

Q1 is performed on the T57 data set to obtain all direct and in-direct sub-classes of a given class. For each T57 data sample, and for each height of the tree hierarchy in that sample, Q1 is executed once using a class in that height. The query time of Q1 is then obtained as the average of the execution times. Q2, Q3 and Q4 are performed on the WordNet inferred data sets. The query time is averaged over 1000 query executions by randomly selecting a WordNet constant to replace the random constant in the above queries. The whole result is shown in Fig.8.

Both axis of Fig.8 is in log10 scale. Lines in Fig.8 are in linear trend, especially Q1. Linear regression analysis of the four lines shows approximately $O(n^{0.84})$, $O(n^{0.89})$, $O(n^{1.06})$, and $O(n^{1.05})$ time complexity for them. In theory, the worst-case time complexity of querying on a database table with index is $O(n \log n)$. The actual query time also depends on the size of the result set. This test shows that the query time has a strong tendency of linear time $O(n)$ complexity and the query is executed quite speedy.

Although the above experiments shows an encouraging query and inference performance result, they are not very thorough and conclusive and the result is still empirical. In the future, more experiments will be performed to further verify the results obtained here.

7 Conclusion and Future Work

ORIENT is a project to develop an ontology engineering tool that integrates into existing industry tooling environments. In this paper, we have described how we addressed the tool integration and scalability issues in the project. By designing

ORIENT as a set of Eclipse plugins, we make ORIENT morph into the Eclipse platform and achieve both data and UI level integration with other tools. We have showed how we implemented a scalable RDF(S) storage, query, manipulation and inference mechanism on top of relational databases. In particular, we have reported the empirical performance of our RDF(S) closure inference algorithm on a DB2 database. Future work of the ORIENT project includes adding OWL support and better visualization methods, improving the integration with Ecore and potentially XMI, and further optimizing and tuning the performance.

References

1. H.Gennari, J., A.Musen, M., W.Fergerson, R., E.Grosso, W., Crubézy, M., Eriksson, H., F.Noy, N., W.Tu, S.: The evolution of Protégé: An environment for knowledge-based systems development. Technical Report SMI-2002-0943, Stanford Medical Informatics (2002)
2. Bechhofer, S., Horrocks, I., Goble, C., Stevens, R.: OilEd: a reason-able ontology editor for the semantic web. In: Proceedings of the Joint German/Austrian Conference on AI. LNCS 2174 (2001) 396–408
3. Corcho, O., López, M.F., Pérez, A.G., Vicente, O.: WebODE: An integrated workbench for ontology representation, reasoning, and exchange. In: Proceedings of EKAW 2002. LNCS 2473 (2002) 138–153
4. G.Karvounarakis, S.Alexaki, V.Christophides, D.Plexousakis, Scholl, M.: RQL: A declarative query language for RDF. In: Proceedings of the Eleventh International World Wide Web Conference (WWW02). (2002)
5. Wilkinson, K., Sayers, C., Kuno, H., Reynolds, D.: Efficient RDF storage and retrieval in Jena2. In: Proceedings of the first International Workshop on Semantic Web and Databases (SWDB), Berlin,Germany (2003) 131–150
6. Eclipse.org: Eclipse platform technical overview. Technical report, http://www.eclipse.org/whitepapers/eclipse-overview.pdf (2003)
7. López, M.F., Pérez, A.G., Amaya, M.D.R.: Ontology's crossed life cycles. In: Proceedings of the EKAW 2000. LNCS 1937 (2000) 65–79
8. Sure, Y., Studer, R.: On-To-Knowledge methodology - final version. Technical report, Institute AIFB, University of Karlsruhe (2002) On-To-Knowledge Deliverable 18, available at http://www.ontoknowledge.org/downl/del18.pdf.
9. Maedche, A., Staab, S., Stojanovic, N., Studer, R., Sure, Y.: SEAL – a framework for developing semantic web portals. In: Proceedings of the 18th British National Conference on Databases. Volume 2097 of LNCS. (2001) 1–22
10. Lei, Y., Motta, E., Domingue, J.: Design of customized web applications with OntoWeaver. In: Proceedings of the 2nd International Conference on Knowledge Capture (KCAP 2003), FL, USA (2003) 54–61
11. Kogut, P., Cranefield, S., Hart, L., Dutra, M., Baclawski, K., Kokar, M., Smith, J.: UML for ontology development. Knowledge Engineering Review **17** (2002)
12. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: A generic architecture for storing and querying RDF and RDF Schema. In: Proceedings of the 1st International Semantic Web Conference (ISWC02). LNCS 2342 (2002) 54–68
13. Hayes, P., McBride, B.: RDF Semantics. W3C Recommendation, W3C (2004) http://www.w3.org/TR/2004/REC-rdf-mt-20040210/.
14. Forgy, C.: Rete: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence **19** (1982) 17–37