Towards Composite Semantic Reasoning for Realtime Network Management Data Enrichment

John Keeney, Liam Fallon
Network Management Lab,
Ericsson,
Athlone, Ireland
{John.Keeney, Liam.Fallon}@ericsson.com

Wei Tai
Predictive Analytics,
SAP Business Objects,
Dublin, Ireland
Wei.Tai@sap.com

Declan O'Sullivan
ADAPT Centre, Computer Science and Statistics,
Trinity College Dublin,
Dublin, Ireland
Declan.OSullivan@cs.tcd.ie

***Abstract*— Monitoring the massive volume of data streaming from managed nodes in Telecommunication networks reacting in a timely manner is increasingly critical for modern Telecommunications Operations Support Systems (OSS). Given the large number and the varieties of the nodes in a telecoms network, the streaming monitoring data is naturally diverse and the volume is often at scales of multiple millions data points each second. These data are well modelled using formal syntaxes (e.g. Management Information Bases), making formal semantics and automated reasoning a viable solution for Telecom data modeling and correlation. This paper proposes an approach that will leverage recent developments in Semantic Reasoning and Big Data. The paper introduces how we propose to use RDF stream reasoning methods for real time event correlation, combined with MapReduce technologies in order to decentralize the large number of reasoning and correlation tasks that need to be undertaken in real time. The proposed approach is currently being implemented and will be evaluated using the diverse data types and volumes that are expected.

I. Introduction

Modern Telecommunications Operations Support Systems (OSS) must deal with massive volumes of performance monitoring data streaming from managed nodes in the network. From this data the OSS must, amongst other tasks, analyze and interpret the current state of each node, the end-to-end status of the network, the connections hosted by the network, the services using those connections, and users using those services. In many cases this incoming data presents as streams of event messages, often at scales of multiple millions of events per second. These events are of different types, from different network elements, with complex data structures and contents. However, one important characteristic of these events is that they are well modelled and formatted, where these models and formats are often subject to formal standards definitions.

In [1] we demonstrate the use of semantic information processing to assess the Quality of Experience (QoE) for individual users using over-the-top (OTT) services hosted by a managed network. Semantic modelling allows the structure, meaning and references to models to be captured using ontologies [2], which can then be executed. Operations can be carried out on models directly using semantic frameworks, while queries and rules can be run on models, and reasoners can automatically deduce knowledge and relationships from the current set of knowledge in a model. The principal advantage of using a semantic information processing approach is that with a combination of an expressive knowledge model, instance data referencing that model, and automated reasoning, new information can be inferred and queried from a knowledge base. More flexible and intelligent applications can then be built using the semantic model and the populated knowledge base rather than using the original source metadata and data.

However, performing semantic reasoning over the monitoring data that streams from a telecommunications network is a difficult and resource intensive task. Depending on the expressiveness of the semantic model and the reasoning approach taken, complete reasoning may be impractical or intractable for even small knowledge bases. However, for well-modelled network monitoring data the schema of the data is well defined and relatively static and stable. The expressivity of the schema is also relatively uncomplicated (or can be reduced to a similar less complicated schema), but the schema is often very large, describing as it does the huge number of measurements, counters and values maintained by network elements. Since monitoring data is machine generated and is defined to always strictly conform to its particular schema then there is little need to perform complete exhaustive reasoning and error checking.

II. Background

II.A. Semantic Reasoning for Network Management

Semantic reasoning can be broadly broken into a set of common interdependent tasks:

- **Concept Subsumption:** checks which concepts are subsumed by other concepts i.e. determine which concepts are more general than others, thus building a concept hierarchy;
- **Checking Individuals**: evaluates if a particular individual is an instance of a concept;
- **Realization:** finds all concepts for which an individual is an instance;
- **Instance Retrieval:** finds all instances of a given concept;
- **Individual Consistency:** checks that instances do not violate the constraints given in the concept descriptions;
- **Concept Satisfiability:** finds contradictions in the definition of a concept, i.e. each concept can have instances and these individuals are not also instances of the concept's complement.

Similar reasoning tasks are also required for property/relationship definitions and assertions.

The reasoner (or inference engine) is responsible for performing these reasoning tasks thereby inferring new information (*axioms/facts*) from the semantic model and the set of stated (asserted) facts based on a set of rules or logics. There are a number of common approaches to perform this reasoning [3] for example: using a forward-chaining (production-based)

978-3-901882-77-7 (c) 2015 IFIP

rule engine to materialize new information at load time; using a backward-chaining (goal-based) rule engine/logic program or a query re-writing approach to reformat a query to perform reasoning in the knowledge-base at query-time; extending the query in the application to manually infer new information without reasoning in the knowledge base; or using a tableau based reasoner where queries are rewritten into a set of satisfiability checking queries/rules that can then be used to check if new candidate axioms are correct or not.

II.B. Key Characteristics of network monitoring data

As mentioned above, semantic reasoning for network monitoring data has a number of key characteristics and requirements that can be exploited in any solution design:

- The schema of the data is relatively static and uncomplicated.
- The volume and velocity of the streamed instance data (measurements) is huge and continuous.
- The instance data are machine generated, fully compliant with the schema and relatively error free.
- Most applications consuming reasoned/enriched data will be continuous themselves (e.g. OSS monitoring/reporting application, data summarization and persistence applications, etc.), therefore for those applications, the reasoning must be either continuous or batched.
- Most of the on-line continuous reasoning tasks will focus on enrichment of instance data, with no (or very little) change to the schema.
- Other non-continuous applications (e.g. examining/analyzing historical information, or OLAP applications) may later query persistent data stores, which may require additional expressive reasoning or once-off off-line reasoning tasks to be performed.

In [1] we present an approach to model network and service monitoring data in a way it can be easily partitioned for standalone processing. The temporal partitioning is achieved according to two temporal aspects of domain concepts; their *dynamicity* and their *temporality*. If the individuals of a concept can only be changed by configuration, the concept has *static dynamicity*; a concept has *dynamic dynamicity* if its individuals are changed at run time by a system using the ontology. If individuals of a concept are not created continuously or periodically during execution, that concept has *global temporality*; individuals of concepts created continuously or periodically have *snapshot temporality* and have an associated timestamp. New incoming instance data (*dynamic, snapshot*) is partitioned into snapshot buckets (windows) according to their timestamps, where these snapshots can then be individually reasoned by combining pre-reasoned fully materialized versions of the of the *static* domain model and *global* configuration. As each snapshot is queried and used by the monitoring application *global dynamic* information can be incorporated into the global model. This careful approach of separating the reasoning tasks into a-priori and a-posteriori tasks, where snapshots do not interact with each other, means that runtime reasoning tasks can be restricted to high-volume limited-expressiveness operations, which in turn can be performed as parallel tasks. However, even this approach does not take into account many of the advantageous feature of streamed network monitoring data as outlined above.

II.C. Event Correlation reasoning over Streams

There exists a number of systems that extend SPARQL for querying streams, where the SPARQL-based syntax is then useful to express event correlation rules. C-SPARQL [4] implements queries that are registered with the C-SPARQL engine and are periodically executed against a stream using a window-based approach to convert RDF streams into chunks of RDF triples *(subject,predicate,object – s,p,o)* with timestamps falling inside a time window. The window is then the target for the query, where the frequency of query evaluations is specified by the time step. CQELS [5] is very similar to CSPARQL but with improved stream support, however like CSPARQL has now inbuilt support for temporal operators for specifying temporal relationships of events. EP-SPARQL [6] offers temporal joining operators such as *SEQ (LEFT) JOIN* and *EQ (LEFT) JOIN*. These operators enables joining between RDF triples to be evaluated not just by finding matches in the *(s,p,o)* triple pattern positions but also by checking if the timestamps of two triples satisfy the given condition. EP-SPARQL is implemented in a semantic Complex Event Processing system ETALIS [7]. It features a rule-based inference engine for event correlation and a set of event descriptions where complex/composite events can be temporally described as correlations of atomic events.

In [8] a number of RDF stream processing systems are compared, including C-SPARQL, ETALIS and CQELS. Unfortunately, none of them shows adequate scalability to satisfy the needs of realtime event correlation for telecom event streams. The results suggest that when the number of queries increases (step by step to 512 queries) the throughput of all three systems decreases drastically. Other work is exploring the use of hardware to improve the performance of stream processing, for example [8] describes an approach for processing RDF streams using GPUs, however these approaches are not applicable to a telecom scenario.

II.D. Improving scalability of semantic reasoning

There is also work that leverages developments in big data technologies to improve the scalability of semantic reasoning. MapReduce can be used to implement distributed reasoning, as shown by Urbani et al. [10]. The basic idea is to convert the join operation in evaluating rule triple patterns into a sorting problem. In the simplest cases, a reasoning rule might be composed of a join operation using two triple patterns based on a join variable as specified in the antecedent (left-hand-side) part of the rule, which generates a set of triples specified in the consequent (right-hand-side) of the rule. This simplest case join operation can be performed in a single iteration of a MapReduce process: the map phase generates a key/value pair for each RDF triple using the join variable value as the key and the entire triple as the value; the reduce phase then generates the inferred triples according to templates in the right-hand-side of the reasoning rule. Therefore finding the joinable triples is then converted to a task of sorting according to key values, which is often natively performed by the underlying MapReduce framework. Urbani et al. also proposes some optimizations to improve performance, including: (1) *Loading Schema Triples in Memory*, (2) *Data Grouping to Avoid Duplicates* which groups triples that can be reasoned in

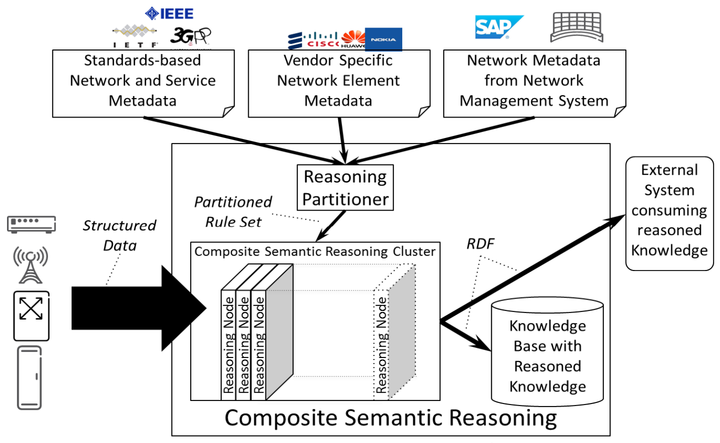
Standards-based Network and Service Metadata
Vendor Specific Network Element Metadata
Network Metadata from Network Management System
Reasoning Partitioner
Partitioned Rule Set
Structured Data
External System consuming reasoned Knowledge
RDF
Composite Semantic Reasoning Cluster
Reasoning Node
Reasoning Node
Reasoning Node
Reasoning Node
Knowledge Base with Reasoned Knowledge
Composite Semantic Reasoning

Figure 1. Proposed approach: Composite Semantic Reasoning for Realtime Network Management Data Enrichment

the same way into an giant intermediate triple to avoid overhead of network traffic and the reduce operation, and (3) *Ordering the application of the RDFS rules*, which categorizes RDFS rules into non-interfering subsets that only need to be processed once (avoids iterative application of rules to reach a fixpoint). The WebPie reasoner [11] uses this approach.

The work in [12] also introduces a concept termed *Transfer Reasoning Tree* where ontology-specific reasoning trees are generated based on RDFS semantics. This approach partitions the ontology schema into subsets of statements that all use one of a few OWL schema properties (*rdfs:subPropertyOf*, *rdfs:subClassOf*, *rdfs:domain* and *rdfs:range*), and generates a set of directed graphs using just these statements. These graphs/trees are then processed against the instance data in separate Hadoop nodes, with limited or no overlap/interference between the nodes. Since this work uses only a subset of RDFS semantics, the reasoning is complete for those semantics but the resulting reasoning is very limited. Such an approach is much more difficult for more complicated semantics where reasoning tasks overlap or interfere with each other.

III. Proposed Approach

To build the system as shown in Figure 1 a number of pragmatic choices can be made based on the requirements for a semantic reasoning approach that would enable real time event correlation of network monitoring data, as outlined in sections I & II. Firstly, since the schema does not change often and is unlikely to be changed by instance data, but will be continuously used as instance data is processed, the schema can be pre-reasoned and fully materialized at load-time using an expressive reasoner such as Pellet [13] or Hermit [14]. Secondly, for off-line applications, reasoner selection can depend on the particular application [11][12][15]. Finally, for on-line data enrichment, where there will be little or no change in the expressiveness requirements of the reasoner, a reasoner configuration can be statically 'composed' specifically for the task. With reasoner composition resource usage and reasoning times can be significantly reduced if the reasoner can adjust itself (e.g. ruleset or reasoning algorithm) based on the characteristics of the ontology to be reasoned, as shown in [3]. Performance can be further enhanced by minimizing the reasoner to remove reasoning tasks that are complex or not strictly necessary to achieve 'just-enough' reasoning completeness.

To compose a reasoner for a specific reasoning task requires a reasoning approach that supports decomposition and reassembly. When the different reasoning approaches are considered the tableau approach emerges as the least-composable due to the inherent tight binding between the reasoning tasks. Therefore, for the continuous runtime reasoning tasks for streamed monitoring data Tableau reasoners are not considered further, however it should be noted that these reasoners are most expressive and are ideally suited to full load-time reasoning and materialization of the schemas to act as read-only input into the continuous reasoning tasks. For continuous reasoning tasks for instances data (*ABox*), where the outputs may be used continuously with limited or no further reasoning (e.g. continuous reporting), a forward-chaining (production-based) rule engine is most applicable. For example, in [17] a ruleset is introduced that does not make any changes to the schema (*TBox*). For data that will later be queried or reprocessed (e.g. OLAP), a backward-chaining (goal-based) rule engine or logic program, or a query re-writing approach is most sensible. A notable exception is [18] which uses an iterative parallel distributed reasoning approach to full materialization without using a forward-chaining production-based rule engine.

In our work, however we primarily focus on a forward-chaining rule engine approach. There are two main complementary approaches to achieve distributed parallelized reasoning using forward-chaining rule engines: 1) partitioning the dataset into subsets across different reasoning engines and then merging the results (e.g. [19] [20]), or 2) partitioning the ruleset into subsets for different reasoning engines and then merging the results (e.g. [21]). Since these approaches are largely orthogonal to each other, a hybrid approach is also possible, using subsets of the dataset and subsets of the ruleset. For complex/expressive datasets using these approaches it could prove difficult to calculate closure of the sub-rulesets and/or sub-datasets. However, since (as explained earlier) the monitoring data coming from managed networks will exhibit as streams of discrete well formed measurements, metrics or counters, we envision that the datasets in question are inherently suitable for partitioning while largely maintaining closure, with a temporal and/or topological partitioning schemes showing most promise, as already shown in [1].

II.A Partitioning the rule set for parallel reasoning

Most forward chaining rule engines are implemented using the RETE pattern matching algorithm [22]. Semantic reasoning rules are defined as a conjunction of triple patterns containing variables (or functions applied to those variables) in the antecedent (left-hand-side) part of the rule, which when matched and joined form combinations of the variables that are used to instantiate and populate a set of triple templates in the consequent (right-hand-side) part of the rule (i.e. when the *l.h.s.* patterns are matched a set of triples are generated from the *r.h.s.* templates). These new triples are then output and inserted into the rule engine memory, then the matching process continues until no new triples are generated.

As shown in [3] for a modest sized dataset and a given ruleset it is possible to determine exactly which rules are required for that dataset, i.e. a cursory examination of the axioms used in the *r.h.s.* of a rule can be used to remove rules for dataset where those axioms are not used (e.g. if the *owl:TransitiveProperty* term is not present in the dataset or the *r.h.s.* of any selected rule, any rule with a *l.h.s.* filter requiring the *owl:TransitiveProperty* term cannot be matched, so the rule will never fire, and the rule can be removed). The same approach can be applied for live-streamed network/service monitoring data. Here the format and model of the data will be well known and will very rarely change. The semantic complexity of this data will also be low so it will be possible to a-priori remove rules that cannot be fired/matched.

If the *r.h.s.* (output) of a rule *r1* contains a term that is present in the *l.h.s.* (input) of another rule *r2*, then output from *r1* combined with other facts in the rule engine memory may match some or all of the input filters of rule *r2*, thus potentially causing it to be fired. This creates a dependency relationship between the two rules (*r1,r2*), where the dependency relationships between all rules in a ruleset can be represented a directed dependency graph. If that dependency graph can be partitioned into unconnected subgraphs then the rule partitions represented in those unconnected subgraphs cannot cause rules in other partitions to be fired. Therefore each subset of the partitioned ruleset can be executed independently in a separate rule engine instance. The results can then be merged in a single operation, ideally suited to a single pass MapReduce process similar to that shown in [11] and [12]. This approach is suitable for any ruleset, including application-/domain-specific rules.

For any complicated ruleset it is unlikely that a clean partitioning of the ruleset can be achieved while maintaining acceptable load balancing between rule engine instances. However we envision that a number of existing strategies to address this can be employed, including: 1) relax requirements for complete/sound reasoning, assuming that later applications consuming the data will perform additional complementary reasoning; 2) allowing multiple/continuous iterations of result merge operations where this process will likely complete after a small number of iterations (depending on the semantics of the data); 3) support overlapping ruleset partitions where individual dependent rules are present in multiple partitions, thus reducing the number of merge iterations required; and 4) where the format and semantics of the data is known and only seldom changes, then pre-runs of the reasoning process can be used to analyze partitioning strategies before being deployed.

II.B Partitioning the dataset for parallel reasoning

As mentioned another complementary approach to parallelized load-balanced reasoning is to partition the dataset as shown in related work. In [18] an approach to parallel reasoning is presented where multiple fully functional reasoner instances process a subset of the dataset to achieve eventual complete reasoning. A scoring mechanism is used to determine which triples are most useful for further sharing and inferencing in other processes. In [23] it is demonstrated that non-schema data (*ABox*) materialization can be fully parallelized for arbitrary partitions of the dataset to achieve RDFS-level reasoning in one partition-merge pass. In [24] a dataset partitioning algorithm is presented where triples (*s,p,o*) with the same subject (*s*) or object (*o*) are maintained in the same partition thus ensuring triples that are likely to have a dependency on each other maintain a degree of locality, and where this partitioning cannot be achieved efficiently then identification of the *other* relevant partitions for a relevant triple can be performed using indexing. In [19] and [20] an approach to partitioning a non-schema (*ABox*) dataset is presented where the data is first divided into *chunks* depending on the subject (*s*) of each triple, with relationships (references to other terms) represented as dependencies between chunks. Chunks are then merged and partitioned (with overlapping chunks appearing in both partitions) to achieve suitable partitioning with minimal interdependencies. These approaches show the benefit of (*s*) subject- and (*o*) object-aware partitioning of non-schema (*ABox*) instance data. When the dataset is derived from streamed network/service monitoring data then a combination of spatial-/topological-based partitioning can be easily achieved since data about different nodes or users will manifest as axioms with user-/node-specific subjects (*s*) or objects (o). This can be further improved when temporal-aware partitioning will be simple to achieve when data is streamed live from relevant nodes.

IV. Discussion and Conclusion

Existing network management models are usually limited in scope to their specific networking domain, are structural and syntactic, and often represent semantic relationships in formats that are not machine readable such as flowcharts, interworking diagrams, state transition diagrams, or even textual descriptions. A semantic modelling approach allows these models to be better leveraged at runtime to support more intelligent and flexible management tools. A key advantage to supporting such a formal semantic model is the ability to perform automated inference tasks on the model and associated instance data, where these inference tasks are implemented in a semantic reasoner. However, a key stumbling blocks inhibiting the use of a semantic models and semantic frameworks is the overhead introduced by the reasoner thereby motivating research to identify domain- or application-specific properties of reasoning tasks for different applications or domains.

In this paper we have discussed the particular characteristics of semantically enabled network management applications, in particular online monitoring of telecommunications networks and services. We have found that different reasoning tasks can be at least partially performed at different times in the application lifecycle, performed by different reasoners with different semantic expressiveness. In particular runtime instance data from monitored network/service elements have very high volume but limited expressiveness and so an application-/model-specific reasoner can be composed for this runtime task. When combined with ruleset-partitioning, dataset-partitioning, and spatial-/temporal-/topological-aware data snapshot partitioning, there exists excellent potential to parallelize/distribute such reasoning tasks to the extent that very high volume reasoning can be achieved for real-time stream-based network/service management tasks.

The authors are in the process of implementing a proof of concept that demonstrates the approach proposed in this paper, and evaluating with a network monitoring scenario with the expected event diversity and volume characteristics.

Acknowledgement

This study is partially supported by the Science Foundation Ireland (Grant 13/RC/2106) as part of the ADAPT Centre for Digital Content Technology (http://www.adaptcentre.ie/) at Trinity College Dublin.

REFERENCES

[1] Fallon, L., O'Sullivan, D.: “The Aesop Approach for Semantic-based End-User Service Optimization”, IEEE Transactions on Network and Service Management, 11(2), 2014

[2] Allemang, D., Hendler, J.: “Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL”. Morgan Kaufmann Publishers Inc., 2008

[3] Tai, W., Keeney, J., O'Sullivan, D.: “Resource-Constrained Reasoning Using a Reasoner Composition Approach”, Semantic Web Journal, vol 6, 2014

[4] Barbieri, D.F., Braga, D., Ceri, S., Della Valle, E, Grossniklaus, M.: “Querying RDF streams with C-SPARQL”, SIGMOD Rec. 39(1), 2010

[5] Le-Phuoc, D., Dao-Tran, M., Parreira, J.X., Hauswirth, M.: “A native and adaptive approach for unified processing of linked streams and linked data”. International Semantic Web Conference (ISWC'11), 2011

[6] Anicic, D., Fodor, P., Rudolph, S., Stojanovic, N.: “EP-SPARQL: a unified language for event processing and stream reasoning”. International Conference on World wide web (WWW'11), 2011.

[7] Anicic, D., Rudolph, S., Fodor, P., Stojanovic, N.: “Stream reasoning and complex event processing in ETALIS”. Semantic Web Journal, 3(4), 2012

[8] Le-Phuoc, D., Dao-Tran, M., Pham, M.D., Boncz, P., Eiter, T., Fink, M.: “Linked stream data processing engines: facts and figures”, International Semantic Web Conference (ISWC’12), 2012

[9] Liu, C., Urbani, J., Qi, G.: “Efficient RDF stream reasoning with graphics processingunits (GPUs)”, International Conference on World Wide Web companion (WWW'14), 2014

[10] Urbani, J., Kotoulas, S., Oren, E., van Harmelen, F.: “Scalable Distributed Reasoning Using MapReduce”. International Semantic Web Conference (ISWC '09), 2009

[11] Urbani, J., Kotoulas, S., Maassen, J., van Harmelen, F., Bal, H.: “OWL reasoning with WebPIE: calculating the closure of 100 billion triples”, European Semantic Web Conference (ESWC'10), 2010

[12] Liu, B, Wu, L., Li, J., Yang, J.J.: “Exploiting Incremental Reasoning in Healthcare Based on Hadoop and Amazon Cloud”, Semantic Cities Workshop at AAAI Conference on Artificial Intelligence (AAAI’14), 2014

[13] Complexible Inc, The Pellet Owl Reasoner, available at https://github.com/complexible/pellet (2015) via http://complexible.com/ (2015)

[14] ISG group at University of Oxford, The HermiT Owl Reasoner, available at http://hermit-reasoner.com/ (2015)

[15] Tai, T., Keeney, J.; O'Sullivan, D.,: “RESP: A Computer Aided OWL REasoner Selection Process” International Conference on Semantic Computing (ICSC’11), 2011

[16] Bock, J., Lösch, U., Wang, H.: “Automatic reasoner selection using machine learning”,. International Conference on Web Intelligence, Mining and Semantics (WIMS'12), 2012

[17] Hogan, A., Harth, A., Polleres, A.: “SAOR: Authoritative Reasoning for the Web”, Asian Semantic Web Conference (ASWC’08), 2008

[18] Oren, E., Kotoulas, S., Anadiotis, G., Siebes, R, ten Teije, A., van Harmelen, F.: “Marvin: Distributed reasoning over large-scale Semantic Web data”, Journal of Web Semantics, 7(4), 2009

[19] Priya, S., Guo. Y., Spear, M.; Heflin, J.: “Partitioning OWL Knowledge Bases for Parallel Reasoning”, International Conference on Semantic Computing (ICSC’14), 2014

[20] Guo, Y., Heflin, J.: “A Scalable Approach for Partitioning OWL Knowledge Bases”, International Workshop on Scalable Semantic Web Knowledge Bases (SSWS’06), 2006

[21] Motik, B., Nenov, Y., Piro, R., Horrocks, I., Olteanu, D.: “Parallel OWL 2 RL Materialisation in. Centralised, Main-Memory RDF Systems”, International Workshop on Description Logics, 2014

[22] Forgy, C., “Rete: A Fast Algorithm for the many pattern/many object pattern match problem”, Artificial Intelligence, 19(1), 1982

[23] Weaver, J., Hendler, J.A.:“Parallel materialization of the finite rdfs closure for hundreds of millions of triples,” International Semantic Web Conference (ISWC’09), 2009

[24] Soma, R. and Prasanna, V. K., “A data partitioning approach for parallelizing rule based inferencing for materialized OWL knowledge bases”, ISCA International Conference on Parallel and Distributed Computing and Communication Systems, (PDCCS’08), 2008