Temporal Reasoning on Twitter Stream using Semantic Web Technologies

by

Meng Cui, B.Sc.(Hons)

Dissertation

Presented to the

University of Dublin, Trinity College

in fulfillment

of the requirements

for the Degree of

Master of Science in Computer Science

University of Dublin, Trinity College

September 2014

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Cui Meng

August 29, 2014

Permission to Lend and/or Copy

Acknowledgments

I would like to express my deepest gratitude to my supervisor, Prof. Declan O'Sullivan and my assistant supervisor Dr. Wei Tai for their excellent guidance, patience on teaching me how to research, giving me so much support and so many valuable suggestions. My dissertation is based on their brilliant ideas and great assistances. I would also like to thank my parents, my friends who are always supporting me and encouraging me with their best effort.

Cui Meng

University of Dublin, Trinity College
September 2014

Temporal Reasoning on Twitter Stream using Semantic Web Technologies

Cui Meng, M.Sc.
University of Dublin, Trinity College, 2014

Supervisor: Prof. Declan O'Sullivan
Assistant Supervisor: Dr. Wei Tai

Social media is gradually becoming an important source of knowledge. For example, Twitter, one of the largest social networks for people to share things and catch up with friends, contains enough information generated by millions of users all around the world.

We can take advantages of Twitter data and semantic web technologies to discover potential trends in varieties of industries and much much more.

In this project, we present a novel approach to perform temporal reasoning on real time Twitter stream using Semantic Web Technologies so that we could derive more valuable information from time dimension data on Twitter. Moreover, in order to deal with such high-frequency data, several filter mechanisms have also been implemented to, significantly, improve the performance of the reasoning process.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Social media is gradually becoming an important source of knowledge. For example, Twitter, one of the largest social networks for people to share things and catch up with friends, contains enough information generated by millions of users all around the world.

Semantic Web is believed to be the future generation of World Wide Web, a “Web of data”, where data could be easily shared, reused and processed by machines [1].

Semantic Reasoning is a very important part of Semantic Web Technologies to enable people to infer implied and valuable information from vast explicit data all over the Internet using customized inference rule set [2]. In order to achieve this, a number of rule engine or semantic reasoners have been developed by researchers all around the world.

Stream reasoner and temporal reasoner are two important categories of semantic reasoner. Stream reasoner aims to enable inference on data stream instead of static data file, while temporal reasoner is designed to enable reasoning on time dimension data.

1.2 Motivation

Stream reasoning [7] and temporal reasoning [8] are widely used in semantic web for various purposes, such as Medical Information System and Network Management System and so on. However, the existing reasoners for both of these reasoning categories have their own limitations.

Firstly, take the most powerful tool C-SPARQL as an example, the stream reasoners at the moment could support continuous but simple queries over the RDF (Resource Description

Framework) data streams [3]. However, temporal functors or temporal reasoning are not fully supported in these stream reasoners, which, as a consequence, makes user unable to derive valuable information that based on time dimension relations.

Secondly, there are a number of existing temporal reasoning approaches. Each of them has presented its own way to introduce time dimension content into RDF data, and implemented temporal functors to cover both instant and interval time relations between different events [4] [5]. Although these approaches are widely used in the real world, like Medical Information System or Network Management System, they could only perform temporal reasoning on huge static historical data but are not capable of dealing with real time data stream.

Thirdly, most of the existing stream reasoners or temporal reasoner are very expensive, both in reasoning time and memory usage.

Finally, social media, like Twitter, is gradually becoming an important source of knowledge. The lack of semantic reasoning on Twitter stream provides a great opportunity. This project will present a novel approach to perform temporal reasoning on real time Twitter data stream. Moreover, in order to deal with such high-frequency data, several filter mechanisms have also been implemented to, significantly, improve the performance of the reasoning process.

1.3 Research Question

According to the background and motivations, the research question of this dissertation is addressed as "*How to semantic reasoning to improve real time analysis of high frequency Twitter stream data*", which could be divided into several small challenges:

1. How to take advantages Semantic Web Technologies to reason on real time Twitter data stream?

2. How to extend the semantic reasoner to perform temporal reasoning over Twitter data stream?

3. In order to deal with such high-frequency Twitter data, is there any way that could be used to improve the performance of reasoning process?

4. How much improvement we could get from these optimization mechanisms?

1.4 Contributions

1.4.1 Semantic Reasoning on real time Twitter Stream

Firstly, we implement a JSON-RDF convertor to transfer Twitter data from JSON format to RDF format that could be accepted by the static semantic reasoner. Moreover, we present a window-based approach to extend the reasoner for reasoning on real time Twitter data stream.

1.4.2 Temporal Reasoning on real time Twitter Stream

This project present our approach to perform temporal reasoning on real time Twitter stream by creating our own temporal functors which could cover all the valuable temporal relations between each two pieces of twitter data.

1.4.3 Reasoning Optimization

We designed and implemented two regular expression based Rule Analyzers, and two filter mechanisms built upon them, so that we can delete all the unnecessary RDF triples (subject-predicate-object data format) before the actual temporal reasoning process. According to the evaluation, these optimization approaches could improve both the reasoning time and memory usage up to 90%.

1.4.4 Configurable Reasoner

Each key components of our reasoner could easily be configured through a configuration file, which increases the flexibility. Moreover, we design and implement an intermediate time format, which could also be customized through the configuration file. Therefore, this reasoner is very extensible to reason on other data streams that with different data formats instead of Twitter.

1.5 Outline of the thesis

Chapter 2 discusses the state of the art of Stream Reasoning and Temporal Reasoning

Chapter 3 presents system requirements, system concept model, system architecture and design decisions.

Chapter 4 provides information about development environment, libraries used, and key components' implementations.

Chapter 5 evaluates our system performance, and the reasoning accuracy.

Chapter 6 concludes our contributions and discuss about the potential future work.

Chapter 2

State of the Art

2.1 Introduction

Social medias, like Twitter or Facebook, are gradually becoming an important source of knowledge. Their high-frequency user generated data has great potential value not only for business or industry area but also for research area. One of the great examples would be [6], where drug-related advertisement could be detected from tweets analysis so that not only manufactures could have better pharmacovigilance over those post-market or investigational drugs, but more importantly, this could provide more safety for patients or drug consumers.

This project aims to reason on Twitter's real time stream data using Semantic Web Technologies. Semantic Web is considered as the next generation of World Wide Web [1]. One of its major breakthroughs is to extend current unstructured or semi-structured data to machine-readable data. The Resource Description Framework (RDF) is a well-known framework to describe web resources so that they could be easily shared and reused across different machines. Our approach will convert Twitter's JSON format data into RDF (subject-predicate-object) triples before pass it into semantic reasoner. In order to properly and efficiently perform reasoning process, several semantic reasoners are compared and evaluated.

This project focuses on extending this existing semantic reasoner to not only perform continuous reasoning on real time Twitter stream but be able to perform temporal reasoning as well. Therefore, a number of stream reasoning and temporal reasoning approaches are also studied.

All in all, in order to achieve temporal reasoning on real time Twitter stream, a number of challenges are required to be addressed, such as Semantic Reasoning, Stream Reasoning and

Temporal Reasoning. Each of these challenges will be discussed in detail in the following sections.

2.2 Semantic Reasoning

2.2.1 Semantic Web Technologies

Semantic Web Technologies are the foundation of this project. Figure 2.1 shows its technology stack. Some of the layers will be discussed in details in this section.

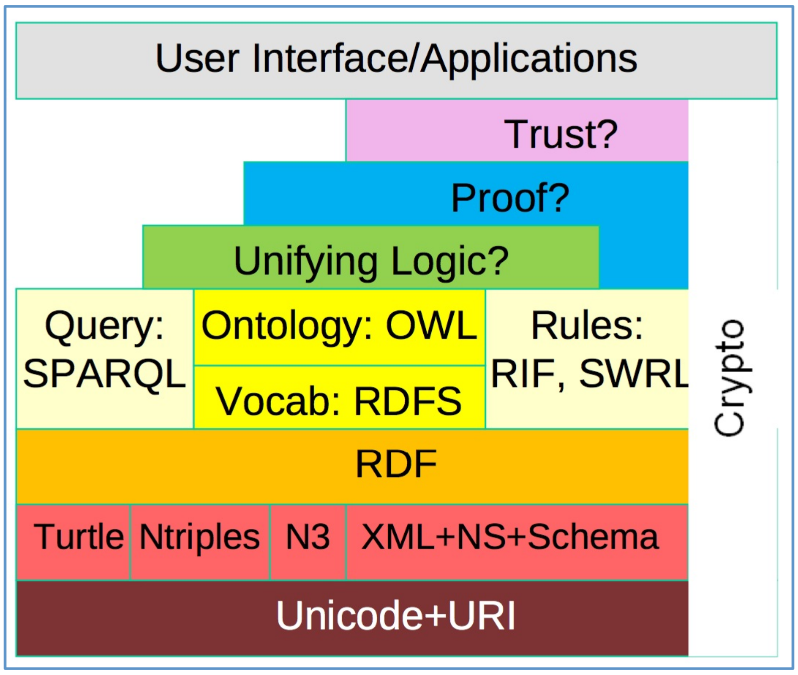


Figure 2.1: Semantic Web Stack[1]

Uniform Resource Identifier (URI): A string of characters to uniquely identify a resource across entire World Wide Web. It always begins with "http", so that the resource identified could be easily accessed through the Internet.

Resource Description Framework (RDF): Different from tree-based XML structure, RDF is graph-based data model to identify things and relations between things using URIs. The core structure of this data model is called triples, each of which contains a subject, a predicate and an object. Each RDF graph consists of a number of these triples.

[1]Copy from Dr. Rob Brennan's lecture notes.

Rules (RIF): Inference is very important in Semantic Web Technology stack to make people reasoning over different data source through rules. Since a number of rule system are used in the real world, World Wide Web Consortium (W3C) working group create Rule Interchange Format (RIF) as a standard to exchange rules between different rule systems [9].

2.2.1 Semantic Reasoners

In order to get better understanding of Semantic Reasoning, several existing approaches are studied.

[10] presents a very detailed description about Jena which is the most commonly used Semantic web toolkit for Java developers. The architecture of Jena and the key structure RDF Graph is well explained which is very valuable for our approach. Moreover, it presents the architecture of Reasoner and explained how different components cooperate with each other to accomplish the inference process. As shown in Figure 2.2, these components are well worth researching before we design and implement our system. In addition, it also present several built-in reasoners, including Transitive Reasoner, RDFS Reasoner and Rubrik Reasoner and their specifics. Rubrik is more valuable to us since it is a rule based RDF reasoner and provide both forward rule engine and backward rule engine for us to use.

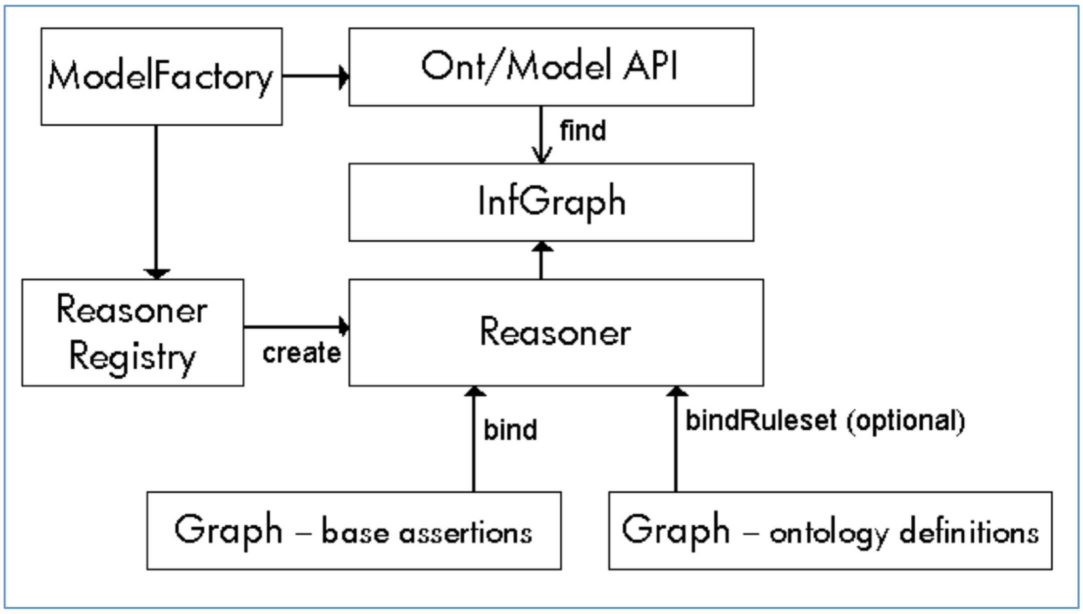


Figure 2.2 Jena Reasoner Architecture from [10]

[10] presents a very detailed description about Jena which is the most commonly used Semantic web toolkit for Java developers. The architecture of Jena and the key structure RDF Graph is well explained which is very valuable for our approach. Moreover, it presents the architecture of Reasoner and explained how different components cooperate with each other to accomplish the inference process. As shown in Figure 2.2, these components are well worth researching before we design and implement our system. In addition, it also present several built-in reasoners, including Transitive Reasoner, RDFS Reasoner and Rubrik Reasoner and their specifics. Rubrik is more valuable to us since it is a rule based RDF reasoner and provide both forward rule engine and backward rule engine for us to use.

[11] presents another production rule based reasoner which is developed to transform OWL ontologies into an object-oriented schema. Instead of RETE rule engine, this approach uses CLIPS to support both rule based and Object Oriented (OO) programming paradigms. In addition, it provides the detailed description and key procedures of **TBOX** reasoning and **ABOX** reasoning, which are also meaningful to our approach. Moreover, it presents an Incremental Rule Loading mechanism, which improves the performance a lot. To provide better understanding, the author not only presents sufficient evaluation result but also explain the theory behind his approach, which is a great way that we could learn to justify our system.

[12] presents detailed design and implementation for rule based reasoner called MiRE4OWL, which is designed to provide reasoning service for resource-constrained devices. In addition, it provides very detailed information about RETE network and explains how it is built with presudocode. Moreover, its lightweight design that results in reducing the memory usage up to 80% could inspire us to optimize our system.

[13] presents a lightweight OWL reasoner called μOR, which is also developed for resource-constrained device. The authors point out that the most commonly used query language SPARQL and its engine are too expensive to run on a resource-constrained device. Therefore, they develop their own language for N-Triples, called SCENT, and their own pattern-matching algorithm called SCENTRA, which are both lightweight. The performance improvement made by μOR is significant comparing to other small reasoners, like Bossam and Pocket KRHyper. Therefore, it is valuable to research on the lightweight language and pattern-matching algorithm implemented.

[14] presents an OWL reasoner, COROR as a new approach for resource-constrained devices, which achieves better time efficiency with less memory cost over other reasoners by implementing two novel algorithms to compose the reasoners on the fly. In addition, this paper presents some brief description about OWL and its sublanguages, and very detailed description about RETE, the fast pattern matching algorithm and presents a number of ways to optimize RETE algorithm. The author also analyzes the advantages and disadvantages of each optimization approach. Moreover, the authors presents very detailed description about the key part of this research, two composition algorithms, i.e. a selective rule loading algorithm and a two-phase RETE algorithm, and also explained two join sequence optimization heuristics they implemented in the beta network construction, i.e. the most specific condition first heuristic and the connectivity heuristic. This research is well studied because our system is developed upon COROR. Therefore it is reasonable to understand the architecture and contributions of this approach.

[15] presents another rule based reasoner for resource-constrained device. This paper is studied because it presents a detailed survey of reasoning approaches on 26 OWL reasoners. The authors classify these reasoners into 5 categories based on their reasoning approaches, including rule-entailment based reasoners, resolution-based reasoners, Description Logic (DL) tableau based reasoners, reasoners using hybrid approaches, and reasoners using miscellaneous approaches. In addition, characteristics of each category are well discussed. This paper provides very helpful information for us to decide which kind of reasoner to use. Since our system need to reason on Twitter stream, the capability of domain-specific reasoning is highly required. Therefore, we choose to build our system on rule-entailment reasoner, as it is very straightforward to model and deploy domain-specific rules.

2.3 Stream Reasoning

Stream reasoning is very widely used in daily life, for example, Medical Information System [23], Transportation System [22] and so on. [7] presents most of the important basics of stream reasoning, including its formal definition and characteristics. It states that **Window** and **Continuous Processing** are the two notions peculiar to stream reasoning. Figure 2.3 shows the window based reasoning approach. For each step, a new window size of data is captured from the data stream and passed to reasoner and in next step, the window moves a certain step forward to capture a new window size of data for the reasoner.

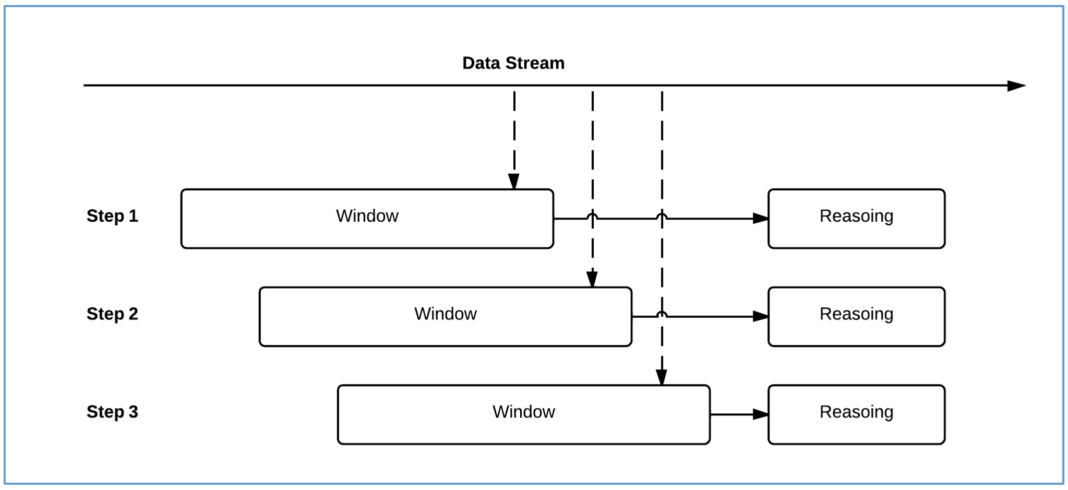


Figure 2.3 Window Based Stream Processing

[7] also presents details about C-SPARQL, which now is a well known and very powerful stream reasoning tool. The author provides very brief comparison between C-SPARQL and its two alternatives, Time-Annotated SPARQL and Streaming SPARQL on their functionalities only and concludes that only C-SPARQL supports continuous processing and aggregation functions.

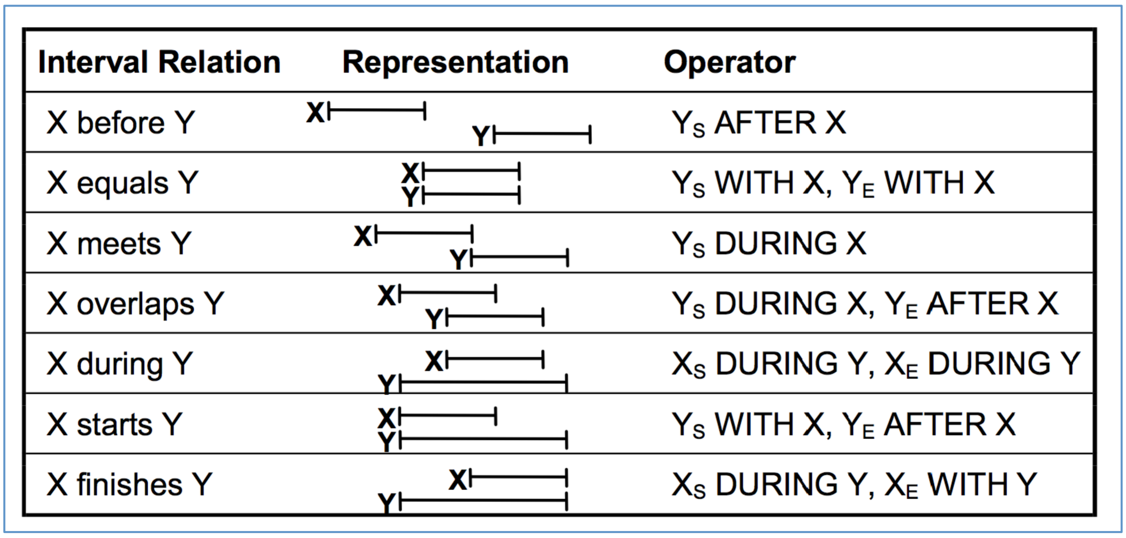
[16] questions a number of challenges faced to combine streaming with semantic reasoning. The data stream management system, which could analyze the changing data during the runtime, could not perform complex reasoning tasks, while reasoners that could perform complex reasoning tasks could not manage reasoning on the rapidly changing information. In addition, the system that could perform reasoning on the changing information could only detect the changes with very limited amount of data and at very low frequency. Therefore, the authors proposed Stream Reasoning as a new approach to integrate Semantic Web technology, data stream and reasoning together to answer the questions. Moreover, the author raised some issues for a stream reasoning system that should be taken into consideration, including limited stream reasoning theory, heterogeneous formats and access protocols, semantic modeling, scale, continuous processing, real-time constraints and parallelization and distribution. The author also presents a discussion of potential measuring progress to evaluate a stream reasoning system, including how fast the data is updated, how many data streams are handled at the same time, how many subscribers are registered, and the time delay between the time that a particular event happened and the time when all the

subscribers got notified and so on. All of these discussions in this paper are very valuable and will definitely inspire the design and implementation of our own stream processor.

[17] introduces Twitter phenomenon at the very beginning to describe the trend that people were caring about feeds and information published on well-known social media which justifies my motivation. In addition, it proposes a novel approach to combine inductive and deductive reasoning with streaming (C-SPARQL) to reduce the gap between streaming and static knowledge analysis. In addition, the author presents a well-designed experiment process to provide better understanding of how their approach will improve the efficiency of C-SPARQL.

2.4 Temporal Reasoning

Temporal reasoning could be easily explained as derive new information from time dimension data. As shown in Figure 2.4, [18] presents the basic interval time relations between two events, which were firstly proposed by Allen in 1980's. This is the start point for almost every temporal reasoning research as specific time relations could be easily designed from it.



Interval Relation	Representation	Operator
X before Y		Y_S AFTER X
X equals Y		Y_S WITH X, Y_E WITH X
X meets Y		Y_S DURING X
X overlaps Y		Y_S DURING X, Y_E AFTER X
X during Y		X_S DURING Y, X_E DURING Y
X starts Y		Y_S WITH X, Y_E AFTER X
X finishes Y		X_S DURING Y, X_E WITH Y

Figure 2.4 Allen's interval time relations (copies from [18])

The authors also present their approach to introduce time dimension relations into Complex Event Processing (CEP) system by design and implement three new temporal functors into their old system. [18] provides me a very clear understanding of what temporal reasoning is and why it is meaningful. With the details of two motivating scenarios, readers could get even better understanding that temporal dependencies could capture more information from the historical events and data. In addition, the paper presented a way to implement temporal operators. With the help of this, researchers could create their own operators to support specific tasks.

More importantly, this paper made me think. Why we really need temporal reasoning or developing temporal operators. If all the historical data is available, could we just write some short program that use file stream reader and simple if-statement to compare the temporal relationship between two events. Yes, we can. However, the truth is people are eager to find a generic way to do the reasoning, more specifically, we need to use the same pattern matching framework to deal with all the rules, no matter it is temporal or not. Therefore, we should implement the temporal operators so that the previous framework could stay the same.

[19] introduces another approach to perform temporal reasoning. It proposes temporal RDF graphs as the main part of a new framework to make pure RDF graphs are capable of handling temporal reasoning. More specifically, they presented a syntax which mainly use both the temporal labels and RDF vocabulary to extend pure RDF graphs to this temporal framework and also presented semantics for temporal graphs, including the definition of temporal entailment.

In addition, it presented a very long and detailed discussion about several issues that might arise when extending RDF with temporal information, e.g. versioning versus time labeling, time points versus time intervals, vocabulary for temporal labeling, temporal entailment and temporal query language. Moreover, it provides detailed description on each of its contributions, including how to define a temporal graph, how to define the temporal semantics and syntax and so on.

[20] presents an approach related to both stream reasoning and temporal reasoning which is motivated by the fact that the state-of-the-art EP only provides the ability of analyzing a large stream of events without considering the background knowledge while separate semantic tools can process static data and analyze context and perform reasoning. The authors of this paper also developed temporal RDF, but they claim their work is different because they tend to detect complex temporal patterns dynamically instead of one time query and response model. The authors also believe more temporal relations should be detected if we want to process more complex reasoning or analysis over the RDF stream. There are researches where all the inference rules are designed before the actually execution and stored in the knowledge base. But the authors think stream reasoning requires to inference processed dynamically during the execution. In my opinion, the automatically generated rules could be a very interesting topic to research on.

2.5 Chapter Summary

In this chapter, we discussed state of the art in Semantic Reasoning, Stream Reasoning and Temporal Reasoning. We compared different reasoners and decided to base our work on rule-based reasoner. In addition, we introduced the basics of stream reasoning and considered the main challenges faced in implementing stream processor. Moreover, we talk about the importance of temporal reasoning, compared several approaches to implement it and finally decided to design and implement our own temporal functors to achieve temporal reasoning. Not all the approaches discussed in this chapter will be used for our development, but the understanding of the pros and cons is important for us the make right decisions.

Chapter 3

Design

In this chapter, we discuss about both the functional and non-functional system requirements, conceptual model of our design and also the system architecture. In the architecture section, we discuss the function of each key component and present several examples to provide readers better understanding of our design. In addition, we present discussions about several valuable design decision-makings and also provide sufficient reasons to prove our decisions.

3.1 Requirements

In order to answer the research questions posed in the introduction chapter and achieve efficient temporal reasoning on real time Twitter Stream, the system should be able to:

1. Capture real time data from Twitter stream.

2. Extend the semantic reasoner to perform temporal reasoning on Twitter Stream.

3. Delete some uninteresting data from the high-frequency Twitter stream to improve the performance of reasoning process.

In addition, there are also some non-functional system requirements that need to be fulfilled:

- **Flexibility:** Each key components of this reasoning system should be easily configured for different purpose

- **Extensibility:** The system should be easily extended to reason on other data stream.

3.2 Conceptual Model

According to the functional system requirements, we present a conceptual model of our design to provide readers basic understanding of how different components work together.

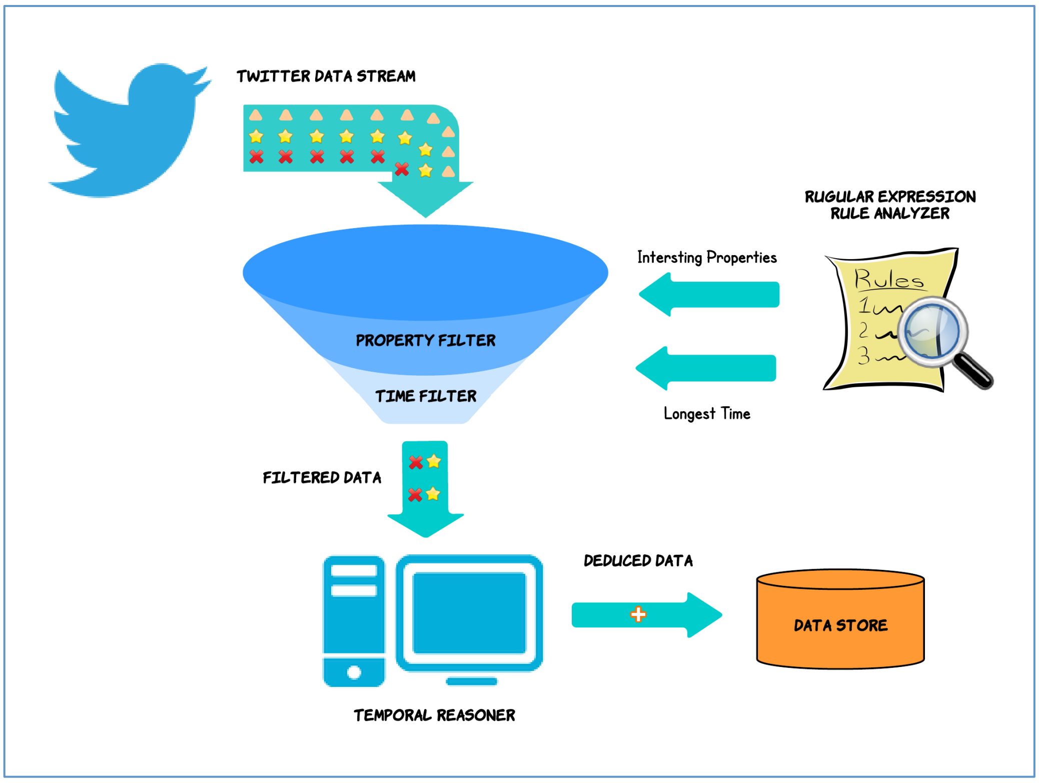


Figure 3.1: Conceptual Model

As shown in the conceptual model, we design two filter mechanisms. One of them is property based and the other one is time based, and both of their filter patterns are based on the analysis of reasoning rule set.

The Twitter data stream will then pass through these two filters before it goes into the Temporal Reasoner where uninterested pieces of data are deleted.

Finally, after the reasoning process, new deduced data will be produced and stored.

3.3 System Architecture

Basic on the conceptual design model, there are four functional parts of the system, including ***Twitter Stream Processing, Rule Analyzers, Filter Mechanisms,*** and ***Temporal Reasoner.***

In this section, we present a more specific system architecture to explain the design of each key components and how they cooperate with each other to match functional requirements.

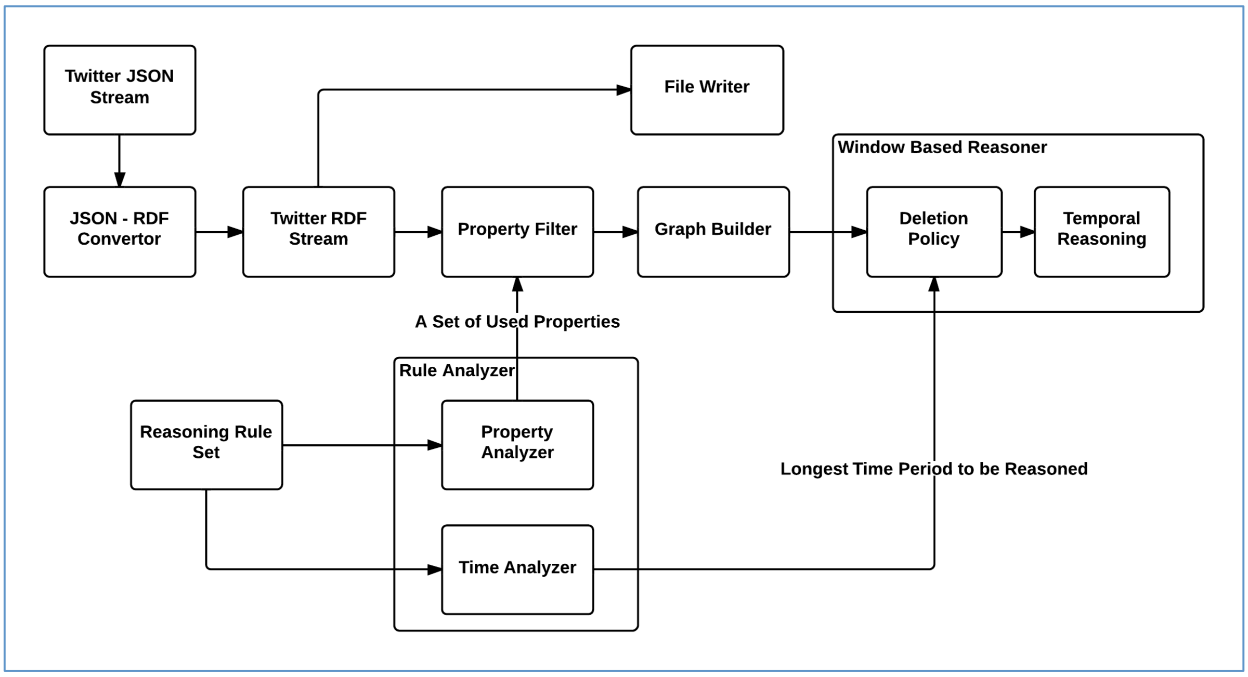


Figure 3.2: System Architecture

3.3.1 Twitter Stream Processing

Twitter Stream Processing is responsible to listen on Twitter Stream, capture real time data and transfer JSON format data to RDF triples. To achieve this, a ***window based processor*** and a ***JSON-RDF convertor*** are designed.

3.3.1.1 Window Based Processor

According to what we discussed in previous section, window based approach is the major method in stream processing. As shown in Figure 3.3, every step the stream processor captures a fixed window size (for example 30 seconds) of data and then passes it to the following components. After each step, the window will more forward in a certain step size (for example

10 seconds), capture new data and pass it again. Therefore, the system is capable of continuous processing of Twitter real time data stream.

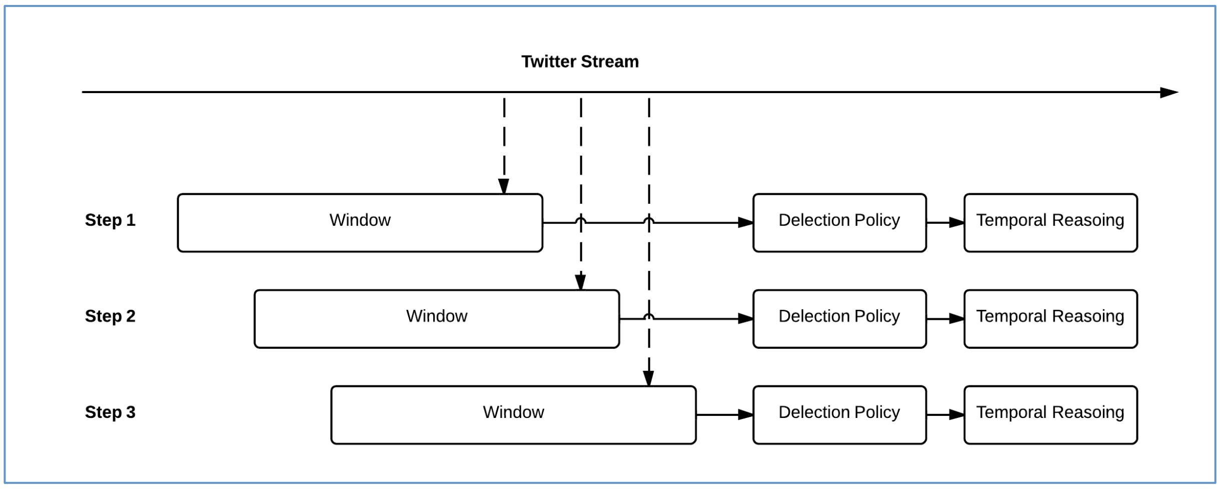


Figure 3.3: Window Based Processor

3.3.1.2 JSON-RDF Convertor

This convertor is designed to accomplish two main tasks.

- Twitter's original data format is JSON, while, in order to perform semantic reasoning, RDF triples are required. Therefore, the convertor needs to convert **JSON data stream** to **RDF triple stream**. N-triples notation is selected to encode RDF triples in this system, because it is a line-based and plain text format, which is very lightweight and easy to construct.

- In addition, as Twitter's data is too complicated, this convertor also selects 17 valuable attributes of full JSON data set, including basic information about each tweet and its relative user.

Figure 3.4 shows the RDF graph for each tweet produced by this convertor, which contains a number of valuable properties, including tweet's text, created time, retweeted count and user's name, friends count, followers count and so on.

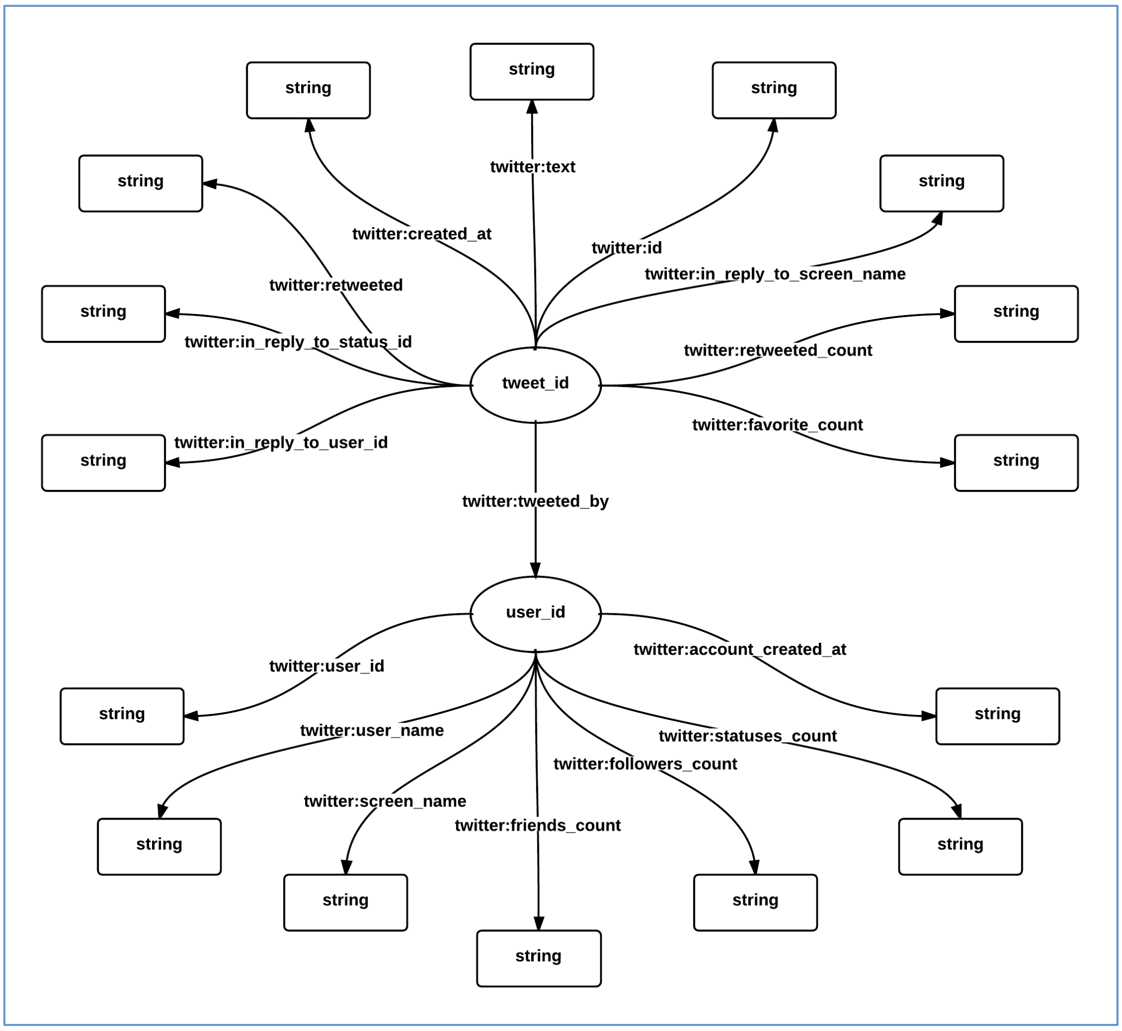


Figure 3.4: RDF Graph for each Tweet

3.3.2 Rule Analyzers

Rule analyzers are the foundation of filter mechanisms and optimization process. There are two main reasons that we design this component:

- Not all rule set needs all of 17 properties of each tweet to perform reasoning.
- Not all reasoning needs the whole window period of data. For example, if the window size is configured to be 30 seconds, while we only interested in the events happened in last 10 second. Then the earlier 20 seconds of data is not required for the reasoning process.

According to these reasons, two kinds of regular expression based rule analyzers are designed to find the characteristics of each reasoning rule set.

3.3.2.1 Property Analyzer

Property analyzer is designed to find all the properties used in a reasoning rule set. Here is an example to demonstrate how this analyzer works.

- **Trend Lead Rule:**

```
(?a twitter:retweeted_count ?r),
(?a twitter:text ?t),
(?a twitter:tweeted_by ?u),
greaterThan(?r, 1000)
->
 (?u twitter:trend_leader ?t)
```

- **Hot Topic Rule:**

```
(?a twitter:retweeted_count ?r),
(?a twitter:text ?t),
greaterThan(?r, 1000)
->
(?a twitter:hot_topic ?t)
```

- **Analyzer Output:**

```
["twitter:retweeted_count",
"twitter:text",
"twitter:tweeted_by" ]
```

As shown in the example, this regular expression based Property analyzer will analyze the whole rule set and return all the used properties in a set. This set will be passed to ***Property Filter***, which will be discussed in filters section.

3.3.2.2 Time Analyzer

Similar to Property Analyzer, this Time Analyzer is also based on regular expression and is designed to find the longest time period that need to be reasoned in a rule set. It only matches interval temporal functors but not the instant ones. There is also an example below to show an example input and output of this analyzer.

- **Temporal Rule 1:**

```
(?a twitter:created_at ?b),
lastXSeconds(?b, 10)
 ->
(?a twitter:happenedInLast10Seconds ?b)
```

- **Temporal Rule 2:**

```
(?a twitter:created_at ?b),
lastXMinutes(?b, 10)
 ->
(?a twitter:happenedInLast1Minute ?b)
```

- **Analyzer Output:**

```
60
```

As shown in the example, this time analyzer will analyze the whole rule set and return the longest reasoning period in seconds. This number will be passed to ***Deletion Policy***, which will be discussed in filters section.

3.3.3 Filter Mechanisms

According to the two rule analyzers presented, we design two corresponding filter mechanisms to delete uninterested data before the reasoning process.

3.3.3.1 Property Filter

Property Filter is designed to delete RDF triples with unused properties based on the analysis result from ***Property Analyzer***. This filter works on the real time Twitter RDF stream produced by ***JSON-RDF Converter*** to delete all uninterested triples before it is passed into ***Graph Builder***.

Graph here is not same as RDF graph. It is a data structure that used in the previous semantic reasoner. Therefore, ***Graph Builder*** is designed as the extended interface of that reasoner to combine a number of RDF triples together as a ***Graph***.

3.3.3.2 Deletion Policy

Deletion Policy is also designed to delete uninterested RDF triples, however, different from ***Property Filter***, it works on the ***Graph*** built by the ***Graph Builder***. As part of the window based reasoner, this rule based ***Deletion Policy*** will firstly be performed on the ***Graph*** to remove out-of-date triples based on the longest period time analyzed by the ***Time Analyzer***, and then the new filtered ***Graph*** will be passed into the final ***Temporal Reasoner***.

Here is an example of the ***Deletion Policy*** rules and the delete functor is designed to delete a triple from a ***Graph***.

- **Sample Rule of Deletion Policy:**

```
(?a twitter:created_at ?t),
NotInlastXSeconds(?t, 10)
 ->
delete (?a twitter:created_at ?t)
```

3.3.4 Temporal Reasoner

In order to achieve temporal reasoning on real time Twitter data stream, a number of temporal functors are designed. In addition, to perform more valuable and more complex reasoning on Twitter stream, several non-temporal functors are also designed.

3.3.4.1 Temporal Functors

Here is a list of temporal functors designed for this system and their descriptions:

- *GreaterThanInstant* (**t1**, **t2**): To compare if time **t1** is greater than time **t2**.
- *LessThanInstant* (**t1**, **t2**): To compare if time **t1** is less than time **t2**.
- *LastXSeconds* (**t**, **x**): To check if time **t** is in last **x** seconds comparing to current system time.
- *LastXMinutes* (**t**, **x**): To check if time **t** is in last **x** minutes comparing to current system time.
- *LastXHours* (**t**, **x**): To check if time **t** is in last **x** hours comparing to current system time.
- *LastXDays* (**t**, **x**): To check if time **t** is in last **x** days comparing to current system time.
- *NotInLastXSeconds* (**t**, **x**): To check if time **t** is **NOT** in last **x** seconds comparing to current system time.
- *NotInLastXMinutess* (**t**, **x**): To check if time **t** is **NOT** in last **x** minutes comparing to current system time.
- *NotInLastXHours* (**t**, **x**): To check if time **t** is **NOT** in last **x** hours comparing to current system time.
- *NotInLastXDays* (**t**, **x**): To check if time **t** is **NOT** in last **x** days comparing to current system time.

These functors could cover both instant and interval time relations from Twitter stream.

3.3.4.2 Non-Temporal Functors

Here is a list of non-functional functors designed for more valuable Twitter data reasoning:

- *IncreasedBeyondX* (**a1**, **a2, x**): To compare **a2 – a1** to check if this value is greater than **x**. This is very useful for some specific scenarios. For example, if we want to find during a certain amount of time which user's followers count has increased beyond 1000, this functor could be used in the rule to compare the difference of two follows counts from the same user with 1000.
- *Delete (* **subject**, **predicate**, **object** *):* As mentioned in previous section, this functor is used in **Deletion Policy** rules to delete expired triples from **Graph**.

3.3.4.3 Intermediate Time Format

In order to design a high extensible reasoning system, an intermediate time format is also designed. Therefore, a new time format could be easily customized from a configuration file and extend this system to reason on new data stream instead of Twitter.

For example, Twitter's time format string is "***EEE MMM dd HH:mm:ss ZZZZZ yyyy***", where ***EEE*** stands for the day of week and ***ZZZZZ*** stands for time zones.

3.4 Example System Flow

In this section, we present a complete example of the whole system to give readers a better understanding of how these key components cooperate with each other to accomplish the whole reasoning process.

There are two things need to be clarified before the example. One is that the time value used in the example like "-15 seconds", which stands for 15 seconds ago, is not the actual time value. This format is used to keep the example short and easy to understand. The other one is that there is only one rule in the reasoning rule set to keep it simple and it aims to find which *new user* has tweeted new things during the last 10 seconds? (New user means twitter user that registered during the last year)

- **Example Reasoning Rule:**

```
(?a twitter:tweeted_by ?u),
(?u twitter:account_created_at ?t1),
greaterThanInstant(?t1, "-1 year"),
(?a twitter:created_at ?t2),
lastXSeconds(?t2, 10)
->
(?u twitter:new_user_new_tweet ?a)
```

Step 1.1 **-** *Property Analyzer Process*: to get all used properties from rule set

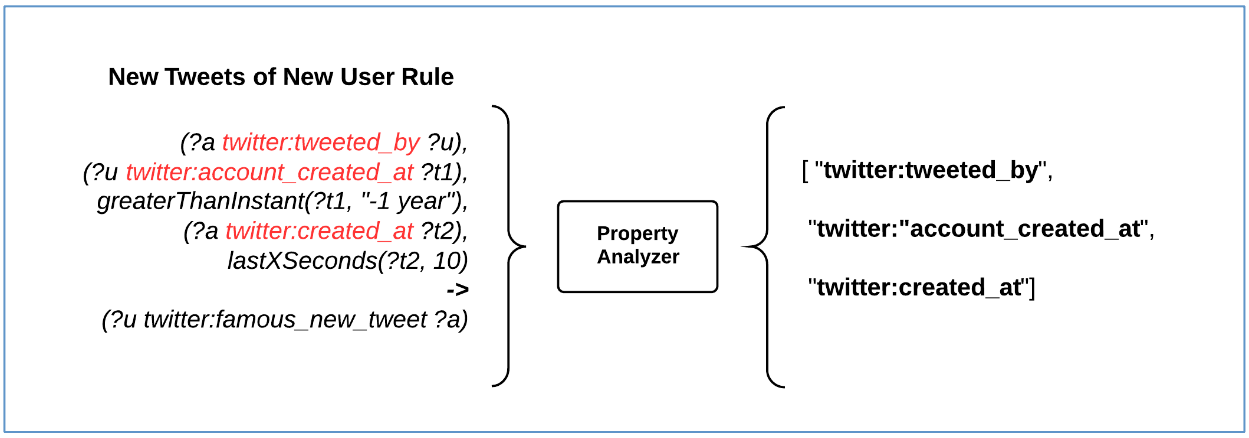

New Tweets of New User Rule
(?a twitter:tweeted_by ?u),
(?u twitter:account_created_at ?t1),
greaterThanInstant(?t1, "-1 year"),
(?a twitter:created_at ?t2),
lastXSeconds(?t2, 10)
->
(?u twitter:famous_new_tweet ?a)
Property Analyzer
["twitter:tweeted_by",
"twitter:"account_created_at",
"twitter:created_at"]

Figure 3.5: Property Analyzer Process

Step 1.2 **-** *Property Analyzer Process*: to get longest period from the rule set

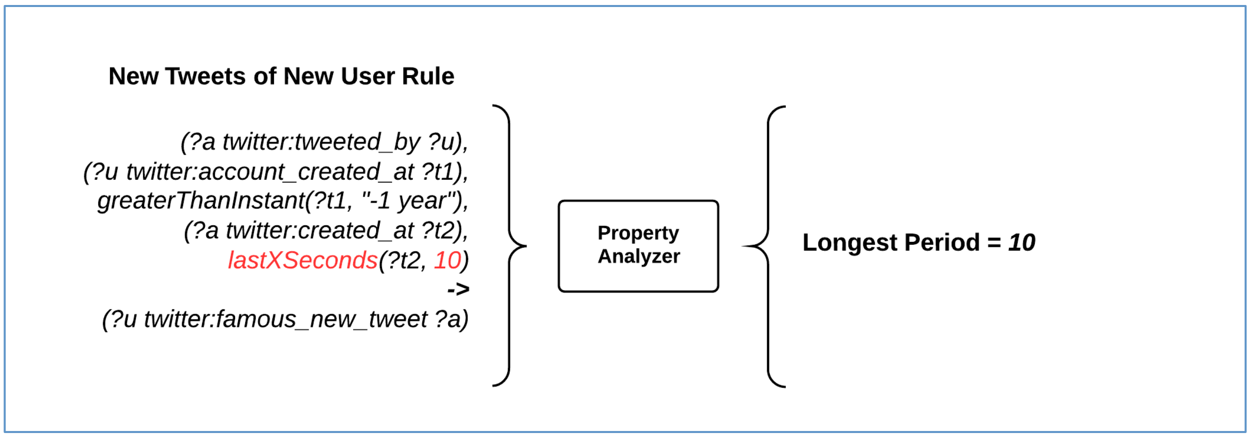
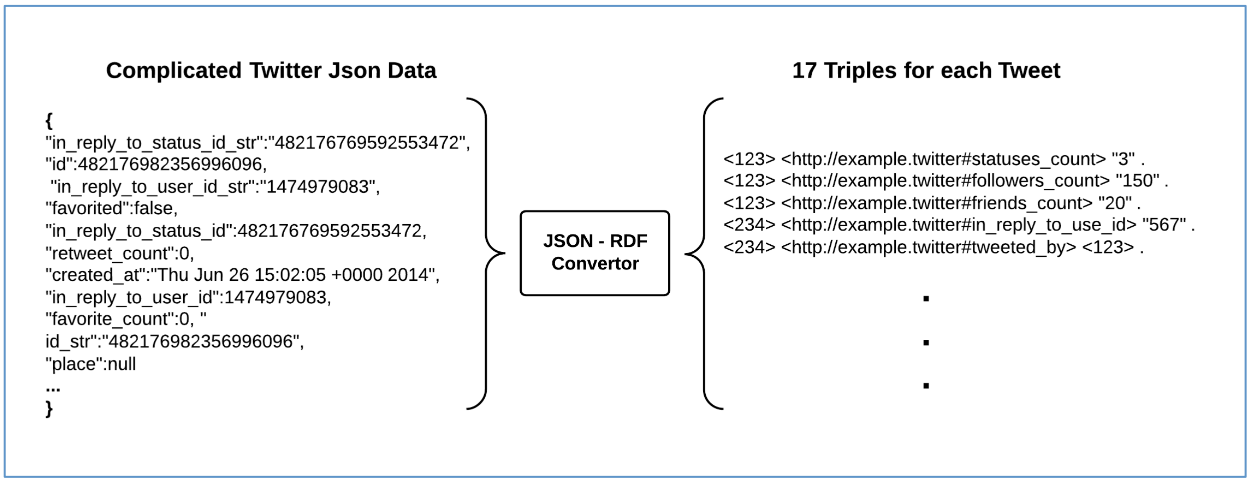

New Tweets of New User Rule
(?a twitter:tweeted_by ?u),
(?u twitter:account_created_at ?t1),
greaterThanInstant(?t1, "-1 year"),
(?a twitter:created_at ?t2),
lastXSeconds(?t2, 10)
->
(?u twitter:famous_new_tweet ?a)
Property Analyzer
Longest Period = 10

Figure 3.6: Time Analyzer Process

Step 2 – *<u>JSON-RDF Convertor Process</u>*: to convert complicated Twitter JSON data to 17 triples for each new tweet.

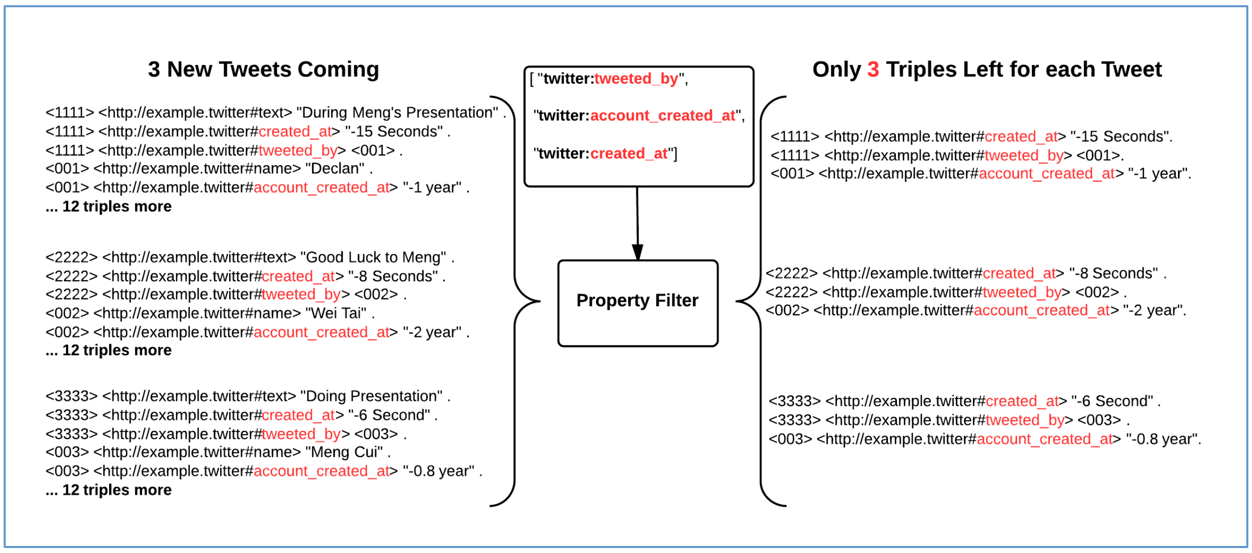


```
{
"in_reply_to_status_id_str":"482176769592553472",
"id":482176982356996096,
 "in_reply_to_user_id_str":"1474979083",
"favorited":false,
"in_reply_to_status_id":482176769592553472,
"retweet_count":0,
"created_at":"Thu Jun 26 15:02:05 +0000 2014",
"in_reply_to_user_id":1474979083,
"favorite_count":0, "
id_str":"482176982356996096",
"place":null
...
}
```

```
<123> <http://example.twitter#statuses_count> "3" .
<123> <http://example.twitter#followers_count> "150" .
<123> <http://example.twitter#friends_count> "20" .
<234> <http://example.twitter#in_reply_to_use_id> "567" .
<234> <http://example.twitter#tweeted_by> <123> .
.
.
.
```

Figure 3.7: JSON-RDF Convertor Process

Step 3 – *<u>Property Filter Process</u>*: to filter on each set of 17 triples to delete the ones with uninterested properties based on the result from ***Property Analyzer***



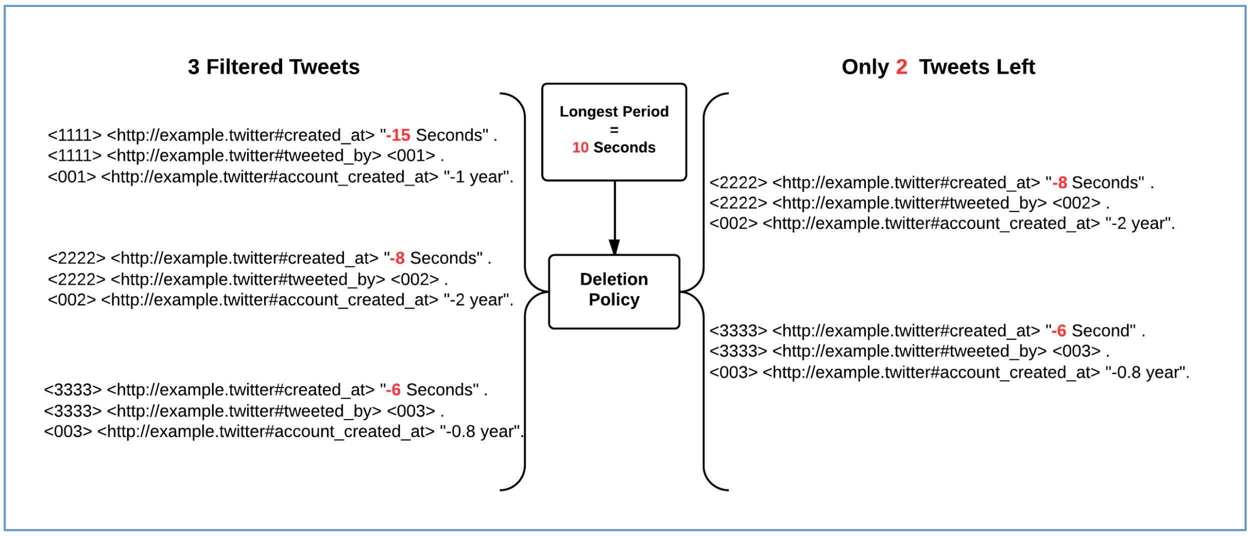
```
<1111> <http://example.twitter#text> "During Meng's Presentation" .
<1111> <http://example.twitter#created_at> "-15 Seconds" .
<1111> <http://example.twitter#tweeted_by> <001> .
<001> <http://example.twitter#name> "Declan" .
<001> <http://example.twitter#account_created_at> "-1 year" .
... 12 triples more

<2222> <http://example.twitter#text> "Good Luck to Meng" .
<2222> <http://example.twitter#created_at> "-8 Seconds" .
<2222> <http://example.twitter#tweeted_by> <002> .
<002> <http://example.twitter#name> "Wei Tai" .
<002> <http://example.twitter#account_created_at> "-2 year" .
... 12 triples more

<3333> <http://example.twitter#text> "Doing Presentation" .
<3333> <http://example.twitter#created_at> "-6 Second" .
<3333> <http://example.twitter#tweeted_by> <003> .
<003> <http://example.twitter#name> "Meng Cui" .
<003> <http://example.twitter#account_created_at> "-0.8 year" .
... 12 triples more
```

```
[ "twitter:tweeted_by",

"twitter:account_created_at",

"twitter:created_at"]
```

```
<1111> <http://example.twitter#created_at> "-15 Seconds".
<1111> <http://example.twitter#tweeted_by> <001>.
<001> <http://example.twitter#account_created_at> "-1 year".

<2222> <http://example.twitter#created_at> "-8 Seconds" .
<2222> <http://example.twitter#tweeted_by> <002> .
<002> <http://example.twitter#account_created_at> "-2 year".

<3333> <http://example.twitter#created_at> "-6 Second" .
<3333> <http://example.twitter#tweeted_by> <003> .
<003> <http://example.twitter#account_created_at> "-0.8 year".
```

Figure 3.8: Property Filter Process

Step 4** – <u>Deletion Policy Process</u>*: to delete expired triples from the ***Graph based on the result from ***Time Analyzer***

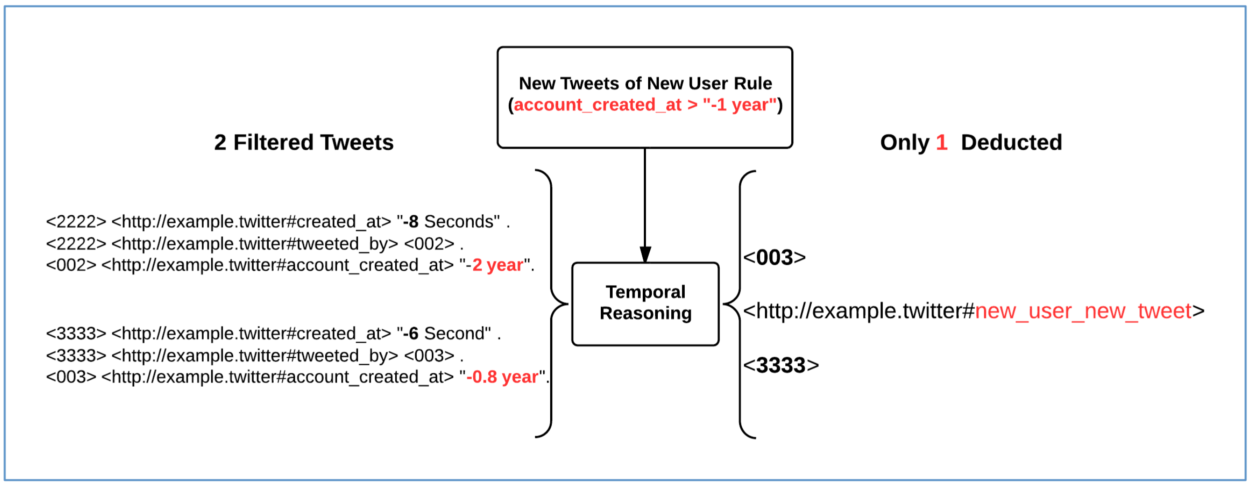


```
<1111> <http://example.twitter#created_at> "-15 Seconds" .
<1111> <http://example.twitter#tweeted_by> <001> .
<001> <http://example.twitter#account_created_at> "-1 year".

<2222> <http://example.twitter#created_at> "-8 Seconds" .
<2222> <http://example.twitter#tweeted_by> <002> .
<002> <http://example.twitter#account_created_at> "-2 year".

<3333> <http://example.twitter#created_at> "-6 Seconds" .
<3333> <http://example.twitter#tweeted_by> <003> .
<003> <http://example.twitter#account_created_at> "-0.8 year".
```

```
<2222> <http://example.twitter#created_at> "-8 Seconds" .
<2222> <http://example.twitter#tweeted_by> <002> .
<002> <http://example.twitter#account_created_at> "-2 year".

<3333> <http://example.twitter#created_at> "-6 Second" .
<3333> <http://example.twitter#tweeted_by> <003> .
<003> <http://example.twitter#account_created_at> "-0.8 year".
```

Figure 3.9: Deletion Policy Process

***Step 5** – <u>Temporal Reasoning Process</u>*: to perform temporal reasoning on filtered data



```
<2222> <http://example.twitter#created_at> "-8 Seconds" .
<2222> <http://example.twitter#tweeted_by> <002> .
<002> <http://example.twitter#account_created_at> "-2 year".

<3333> <http://example.twitter#created_at> "-6 Second" .
<3333> <http://example.twitter#tweeted_by> <003> .
<003> <http://example.twitter#account_created_at> "-0.8 year".
```

```
<003>
<http://example.twitter#new_user_new_tweet>
<3333>
```

Figure 3.10: Temporal Reasoning Process

3.5 Design decisions

A huge number of decisions have been discussed and made to achieve this final system design, while some of them are very valuable and worth being presented here.

1. *Why choose Twitter Stream to reason on?*

   At the beginning of this project, we focused on extending a static semantic reasoner to be capable of stream reasoning and temporal reasoning. However, every concept was too general, so it was very hard to start. Therefore, we decided to think about some scenarios that this project should be capable of which led us to Twitter. Not only because Twitter is popular and familiar with everyone, but also because Twitter's data stream has very high frequency and high diversity, which provided us a lot of opportunities to reason on.

2. *Why choose property related filter and time related filter?*

   In fact, there are a lot of other filters could be designed, but the reason to pick up these two is very sufficient. That is because these two filters are more generic than the others. More specifically, these two filters are not specific for Twitter data reasoning and are not specific to any scenarios. Therefore, they could be easily extended to filter on other streams, as long as they have a number of properties for each RDF graph and they need to reason on time interval relations.

3. *Why design these two filters in different way?*

   As mentioned in previous sections, ***Deletion Policy*** is rule based filter mechanism to delete expired triples from ***Graph***, while ***Property Filter*** is not rule based and it removes the triples from the RDF stream before the ***Graph*** is built. Therefore, the ***Deletion Policy*** is easier to be configured and customized, while ***Property Filter*** is more efficient. In fact, these two filters could be designed in the same way to perform the same functionality. However, we prefer to design them in different ways so that the readers and following researchers would know there are two possible ways to design filters so that they could have more choices to work upon our work or design their own systems.

3.6 Chapter Summary

In this chapter, we present both the functional and non-functional system requirements, conceptual model of our design and also the system architecture. In the architecture section, we present detailed design of each key component and provide sufficient examples to demonstrate how they cooperate with each other. In addition, we also present several valuable design decision-makings, which could inspire the implementation.

Chapter 4

Implementation

In this chapter, we present the programming language and several libraries used, and also provide our development environment as a reference. Moreover, we present major process to implement the key components so that we could fulfill the requirements discussed in the design chapter.

4.1 Programming language and libraries

4.1.1 Java Language

As discussed in the previous chapters, our approach begins with extending an existing semantic reasoner. Therefore, we continue to use the same programming language ***Java*** to accomplish our implementation.

4.1.2 Twitter4j

Twitter4j is not an official library for Twitter API, but it is very widely for Twitter related development, because of:

- **100% pure Java:** It is developed by Java and is compatible for any Java platform version 5 or latter.

- **Built-in OAuth Support**: *OAuth* is an authorization framework that used by Twitter to enable a third-party application to obtain limited access to Twitter API service. By using ***Twitter4j*** library, *OAuth* could be easily configured in a properties file.

- **Zero Dependency:** There is no additional library needed to use ***Twitter4j***

- **100% Twitter 1.1 API supported**

Apart from these technical reasons, ***Twitter4j*** also provide good documentations and sufficient examples to demonstrate the most commonly used functions, which as a consequence, makes it even easier to be used.

4.1.3 Jena

Jena is an open source Java framework for developing Semantic Web applications. It provides a variety of APIs for different functional purpose, including *RDF API*, *SPARQL API*, *Text Search API, Security API,* and *Java Database Connectivity (**JDBC**) API* and so on. For our development, *RDF API* is used in ***RDF-JSON Convertor*** to produce the RDF data stream.

In addition, *JSON API* is also included in Jena library, which could be used to fetch JSON object and exact values from it.

4.2 Development Environment

Specific information of system environments and Integrated Development Environment (**IDE**) are presented in this section as a reference for further development.

4.2.1 System Environment Information

System Component	Description
System Version	Mac OS X 10.8.5
Processor	2.7 GHz Intel Core i7
Memory	16 GB 1600MHz DDR3

Table 4.1: System Environment Information

4.2.2 IDE Information

Environment Component	Description
IDE	Eclipse
Version	Kepler Service Release 1
Java Version	1.6

Table 4.2: IDE Information

4.3 Key Components Implementation

In this section, we present the key steps to implement each key component.

4.3.1 Window Based Twitter Stream Processer

In order to accomplish Twitter stream processer, a data stream listener and a window based buffer need to be implemented.

Twitter Stream Listener

Twitter4j *stream API* is used to listen on Twitter's public data stream and get notification for each new tweet.

Window Based Buffer

This buffer stores a fixed window size of Twitter data and in each new step, a certain time of old data is removed from the buffer and same period of new data is added. According to these requirements, a circular list is implemented to maintain this fixed size buff. Figure 4.1 shows a more intuitive view of how this buffer works.

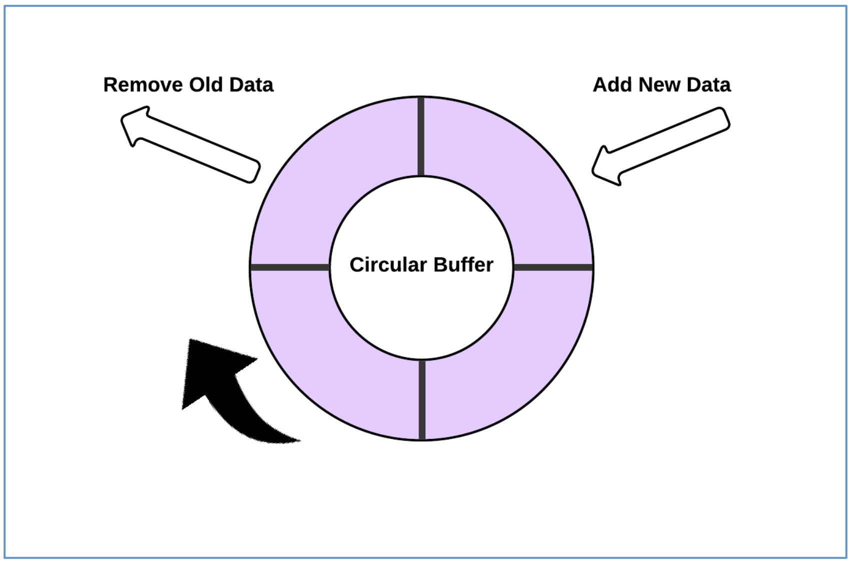


Figure 4.1: Circular Buffer

4.3.2 Rule Analyzers

As discussed in the previous chapter, both the ***Property Analyzer*** and ***Time Analyzer*** are based on regular expression, which is a very commonly used technic in text processing. It defines a number of metacharacters, which could be combined to form a search pattern for specific string matching purpose. Table 4.3 shows some commonly used metacharacters and their meaning in regular expression.

Metacharacter	**Description**
.	Matches any single character
$	Matches the ending position of a string or the ending position of any line.
*	Match the preceding element zero or more times.
?	Match the preceding element zero or one time.
+	Match the preceding element one or more times.
( )	Defines a marked subexpression.
[ ]	Match a single character contained within the brackets.
{ m, n }	Match the preceding element *at least* **m** and *not more* than **n** times

Table 4.3: Metacharacter Descriptions

According to these metacharacters, two search patterns are implemented. One is integrated in ***Property Filter*** and is responsible for matching all the twitter properties appear in the rule set. The other one is integrated in ***Time Filter*** and is responsible for matching all the interval temporal functors in the rule set and comparing to find the longest period need to be reasoned.

4.3.3 Filter Mechanisms

According to our discussion in the previous chapter, the two filter mechanisms are implemented in different ways:

- **Property Filter:** is implemented as code-based filter to process on Twitter RDF stream. For each of the 17 triples in each tweet, it traverse the set of used properties passed from the ***Property Analyzer*** to determine if this triple is needed or not.

- **Deletion Policy:** is implemented as rule-based filter to process on the ***Graph.*** It traverses the whole ***Graph*** to find the expired tweet and delete all the triples belong to that tweet.

4.3.4 Temporal Reasoner

As discussed in the deign chapter, several new functors need to be implemented to achieve temporal reasoning on Twitter stream. In addition, in order to make this reasoner easy to extend to reason on other data streams, an intermediate time format is also implemented.

New Functors Implementation

There are several steps need to follow to implement a new functor:

1. Pick up a simple and meaningful name for the functor, which will be used in reasoning rules.

2. Specify the number of parameters that this functor need.

3. Implement the functor body, which is actually main functional part of each functor. People could implement a variety of body functions based on their own requirements.

4. Register the new functor so that it could be recognized by the reasoner during the rule parsing process.

Intermediate Time Format Implementation

This time format is implemented to read the customized time format string from the configuration file and then use *SimpleDateFormat* Java class to convert each time format to a customized Java *Date* class so that the time values with different format could be comparable.

4.4 Chapter Summary

In this chapter, we presented the programming language and a number of libraries we used, and also provided our development environment as a reference. In addition, we presented the major processes needed to implement each key component, which makes it easy for other researchers to perform further development upon our work.

Chapter 5

Evaluations

After the development, it is very important to evaluate how well our system could perform. In this chapter, we will evaluate not only the system performance but also the reasoning accuracy. Both of these two aspects are very critical to our system, since without significant performance improvements, these novel filter mechanisms will make no contribution to this area; without high accuracy, this approach will not be trusted by other researchers.

5.1 The Rationale

Before presenting the actual experiment, it is reasonable to discuss factors that will affect the system performance and the metrics we select to evaluate the system. We will also provide sufficient reasons to justify our choices.

5.1.1 Factors

There are a number of factors that could affect the system from different aspects. However, since we put our focus on evaluating the performance improvements by these two novel filter mechanisms we presented, only several of these factors are selected.

- **Window Size:** stands for a certain period time to measure the buffer size and also measure how much data is captured from Twitter stream in each cycle. This factor directly determines, in each cycle, how many triples are processed by the system and how many triples are reasoned by the temporal reasoner. In this chapter, window size is measured by seconds (s).

- **Rule Complexity:** stands for the number of different properties appeared in one reasoning rule set. Comparing to number of reasoning rules, this factor has more influence on the performance since the ***Property Filter*** is based on number of properties used in the rule set. In this chapter, the range of **Rule Complexity** is from 1 to 17 since 17 is the total number of properties each tweet has.

5.1.2 Metrics

To present a reasonable evaluation result of the performance improvements that our approach achieved, several metrics are selected:

- **Number of Triples:** is the most intuitive metric to measure how many triples are processed by the system and how many of them are deleted by the filter mechanisms. Therefore, this is a very valuable metric to provide direct result of the system performance.

- **Memory Usage:** is the most commonly used metric to measure system performance. In addition, the result of this metric could be more valuable to other researchers as they could easily compare their approach's performance with ours. In this chapter, this metric stands for the memory usage over the whole reasoning process and is measured in kilobytes (KB).

- **Reasoning Time:** is another commonly used metric. Similar to **Memory Usage**, this metric is also very valuable to other researchers to compare. In this chapter, **Reasoning Time** stands for the reasoning time consumed over the whole reasoning process and is measured in milliseconds (ms).

- **Reasoning Accuracy:** is a very quality metric especially for our system. Since we implemented two filter mechanisms, it is essential to evaluate, apart from deleting the uninterested data, if they also delete some interested data as well. This presents if the filter mechanisms will affect the number of triples deduced. Equation 5.1 shows how this metric is calculated.

$$\textit{Reasoning Accuracy} = \frac{\text{Number of triples deduced with filter on}}{\text{Number of triples deduced without any filter}} \quad (5.1)$$

5.2 Experiments

According to what we discussed in previous section, a number of experiments are designed and conducted to evaluate the system performance from different aspects. *Note that* in this chapter, every data value shown in the experiment result is the **average** value of **20** runs of the same experiment.

5.2.1 Experiments about Number of Triples

5.2.1.1 Number of Triples *vs* Property Filter

Experiment Setup

This experiment collects 6 groups of data with increasing **Window Size** from 10 seconds to 60 seconds, and in each group, there are two values of **Number of Triples** metric. One is collected with no filter on, and the other one is collected with only **Property Filter** on. In this experiment, the **Rule Complexity** of the reasoning rule set is **7**, which is a medium level and longest time period is **10** seconds.

Results

Figure 5.1 shows the evaluation results of this experiment.

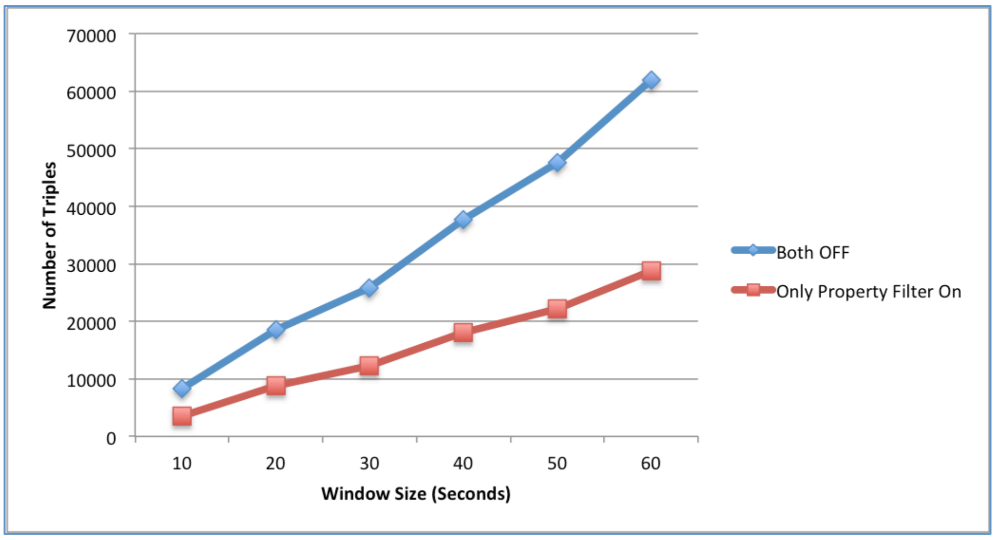


Figure 5.1 Number of Triples *vs* Property Filter

Findings

According to the result, we could find that Property Filter works well on reducing the total number of triples to be reasoned. The improvement is proportional to **Rule Complexity**. As shown in Figure 5.1, the Property Filter could reduce the number of triples by more 50% while the **Rule Complexity** is set to 7, which is also nearly 50% of 17, the maximum value.

5.2.1.2 Number of Triples *vs* Deletion Policy

Experiment Setup

This experiment collects 6 groups of data with increasing **Window Size** from 10 seconds to 60 seconds, and in each group, there are two values of **Number of Triples** metric. One is collected with no filter on, and the other one is collected with only **Deletion Policy** on. In this experiment, the **longest period time** of the reasoning rule set is **10** seconds, which is a reasonable level and the **Rule Complexity** is **7**.

Results

Figure 5.2 shows the evaluation results of this experiment.

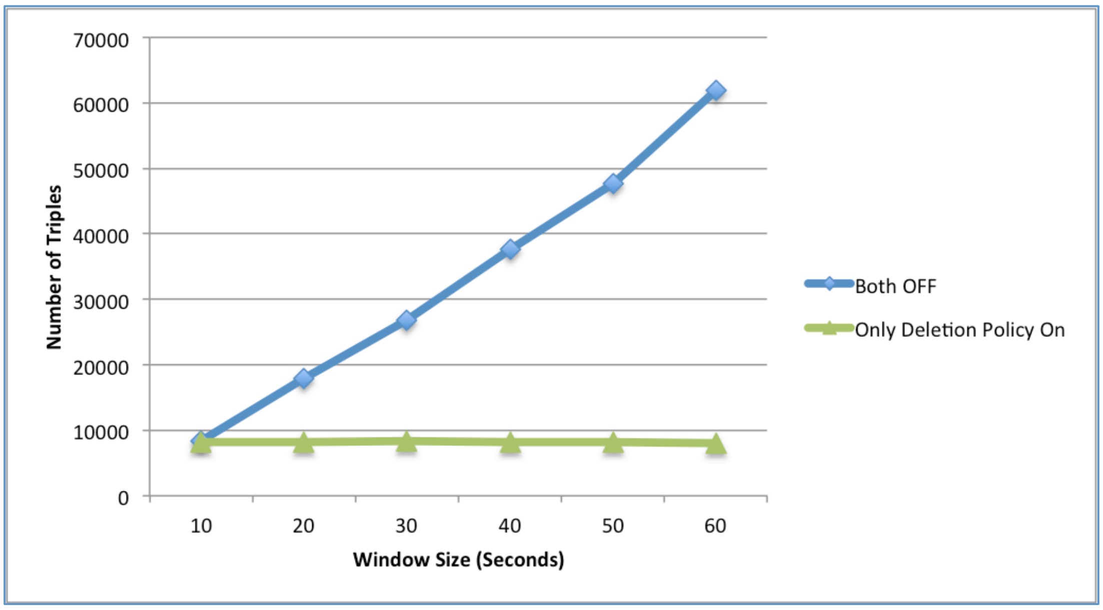


Figure 5.2 Number of Triples *vs* Deletion Policy

Findings

According to the result, we could find that **Deletion Policy** works very well on maintaining the total number of triples to be reasoned in very low level, even with an increasing window size. The level of this flat line is determined by the **longest period time** of the rule set. When the window size is smaller than this threshold, the **Number of Triples** should make no big difference between these two conditions (**Deletion Policy** on or off). Once the window size is greater than this threshold, the **Number of Triples** value for **Deletion Policy** off will still increase, while the value for **Deletion Policy** on will stay in the previous low level.

5.2.1.3 Property Filter *vs* Deletion Policy on Number of Triples

Experiment Setup

This experiment collects 6 groups of data with increasing **Window Size** from 10 seconds to 60 seconds, and in each group, there are two values of **Number of Triples** metric. One is collected with only **Deletion Policy** on, and the other one is collected with both **Deletion Policy** and **Property Filter** are on. In this experiment, the **longest period time** of the reasoning rule set is **10** seconds and the **Rule Complexity** is **7**.

Results

Figure 5.3 shows the evaluation results of this experiment.

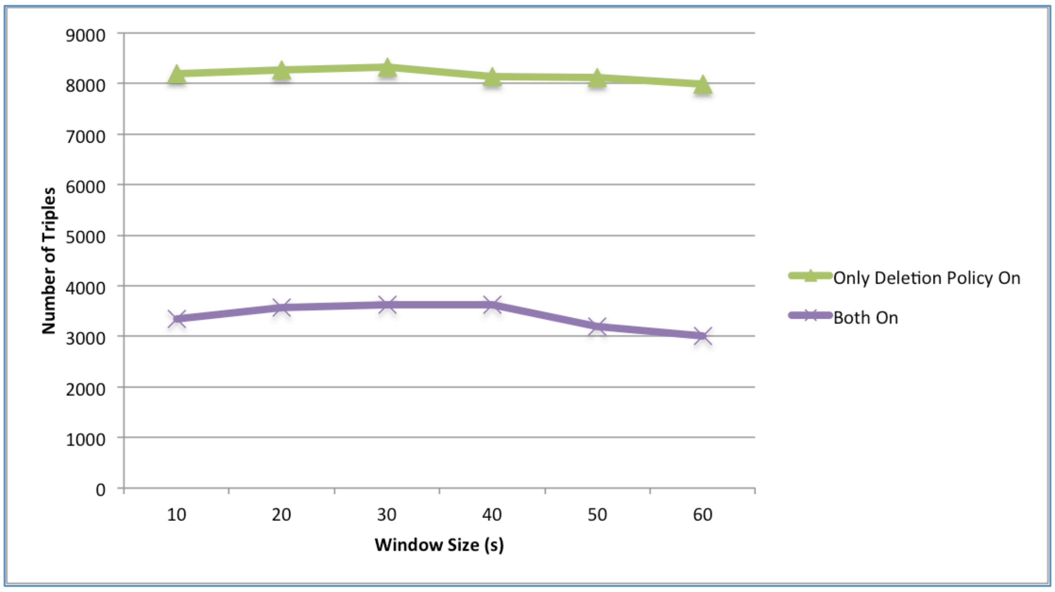


Figure 5.3 Property Filter *vs* Deletion Policy on Number of Triples

Findings

According to the result, we could find that **Property Filter** could make further improvements even though the **Deletion Policy** is already on. This could be easily explained in theory as these two filter mechanisms are designed from different perspectives. One is based on used properties, and the other is based on longest time need to be reasoned. As shown in Figure 5.3, the further improvements made by **Property Filter** is still proportional to **Rule Complexity**, which is very reasonable.

5.2.1.4 Combined Results on Number of Triples

Figure 5.4 shows the combined results of four different conditions, both filters are off, only **Property Filter** is on, only **Deletion Policy** is on, and both filters are on. We present all groups of data in one diagram to provide readers an overall view of how much improvement each filter could achieve, performance comparison between these two filters and different specifics each filter has.

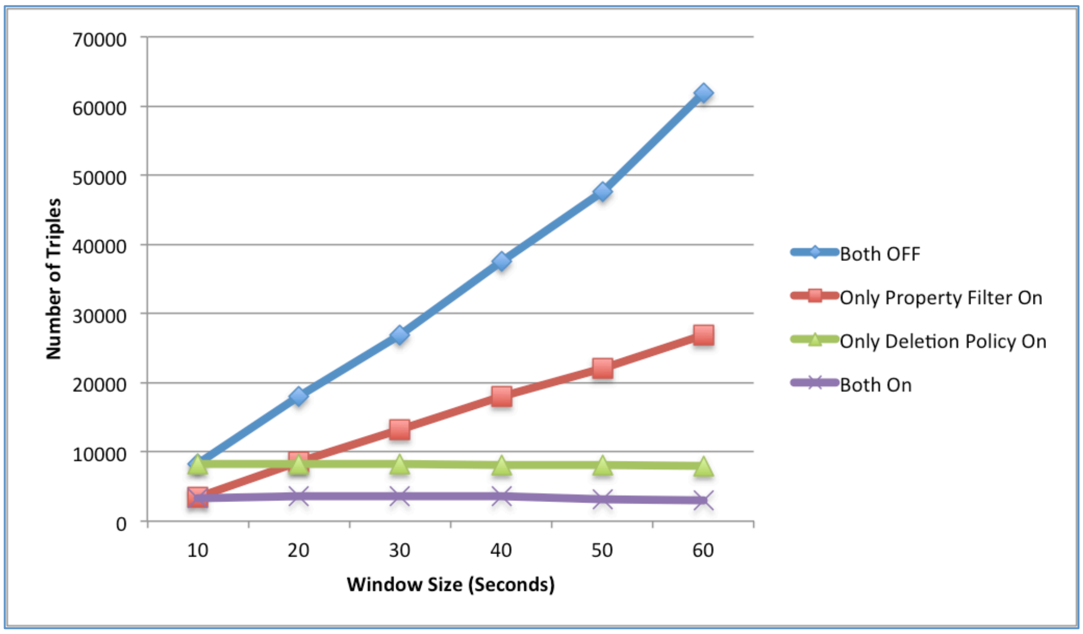


Figure 5.4 Combined Results on Number of Triples

Findings Summary

In this series of experiments, the **Rule Complexity** of the rule set is **7** and **longest time period** is **10** seconds. According to Figure 5.4, we could summarize our findings as:

- **Property Filter** can delete more than **50%** triples, which is proportional to **Rule Complexity**.

- **Deletion Policy** can maintain the total number of triples under a very low level even with increasing window size. However, the level of this flat line is determined by the **longest time period** of rule set.

- **Property Filter** could still make contribution to removing more triples even when **Deletion Policy** is already on. The further improvement made is also proportional to Rule Complexity.

5.2.2 Experiments about Reasoning Time

5.2.2.1 Reasoning Time *vs* Property Filter

Experiment Setup

This experiment collects 6 groups of data with increasing **Window Size** from 10 seconds to 60 seconds, and in each group, there are two values of **Reasoning Time** metric. One is collected with no filter on, and the other one is collected with only **Property Filter** on. In this experiment, the **Rule Complexity** of the reasoning rule set is **7**, which is a medium level and longest time period is **10** seconds.

Results

Figure 5.5 shows the evaluation results of this experiment.

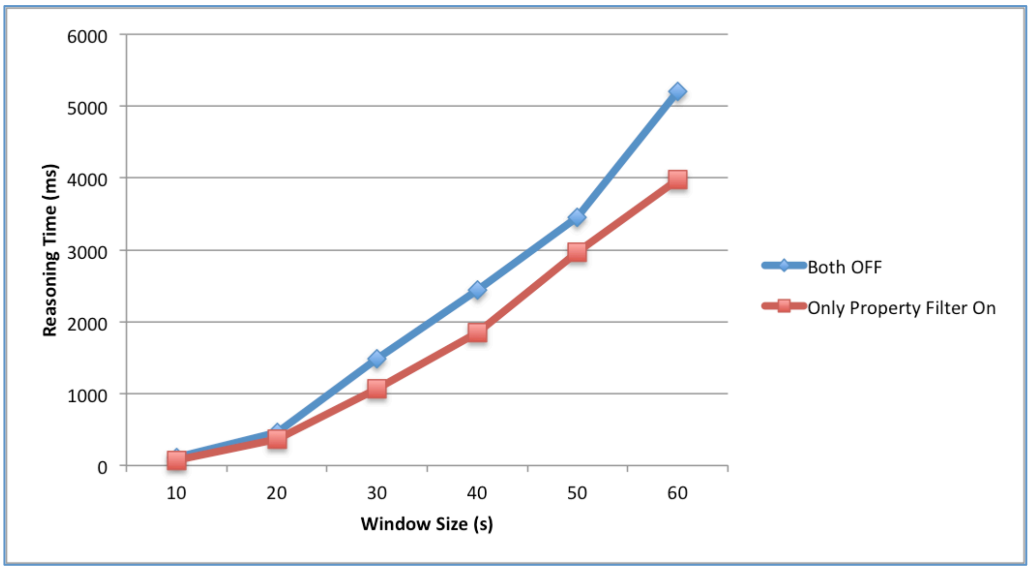


Figure 5.5 Reasoning Time *vs* Property Filter

Findings

According to the result, we could find that Property Filter reduce the **Reasoning Time** around 25%, which is less than its affects on **Number of Triples**. This is because the Reasoning Time also covers some processes that are not only affected by the **Number of Triples**, including RETE network building [21], rule pattern matching and so on. Even though the inference is not as big as it made on **Number of Triples**, 25% improvement is significant enough for a filter mechanism.

5.2.2.2 Reasoning Time *vs* Deletion Policy

Experiment Setup

This experiment collects 6 groups of data with increasing **Window Size** from 10 seconds to 60 seconds, and in each group, there are two values of **Reasoning Time** metric. One is collected with no filter on, and the other one is collected with only **Deletion Policy** on. In this experiment, the **longest period time** of the reasoning rule set is **10** seconds, which is a reasonable level and the **Rule Complexity** is **7**.

Results

Figure 5.6 shows the evaluation results of this experiment.

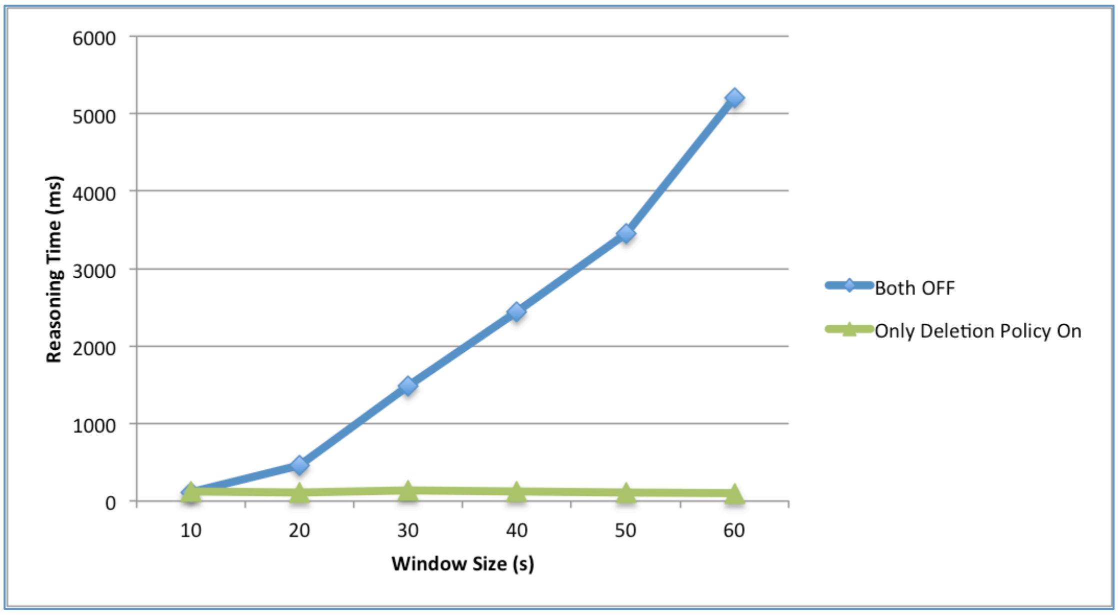


Figure 5.6 Reasoning Time *vs* Deletion Policy

Findings

According to the result, we could find that **Deletion Policy** makes even better improvements on **Reasoning Time** comparing to **Number of Triples**. This is because the **Reasoning Speed** decreases a lot as the total number of triples increases, in other words, the reasoner could process much less triples per second if the total number of triples is getting larger. According to our previous findings, **Deletion Policy** could maintain the **Number of Triples** in a very low level, which as a consequence, resulting in maintaining the **Reasoning Time** in a much lower level. In addition, the level of this flat line is still determined by the **longest time period** of the rule set.

5.2.2.3 Property Filter *vs* Deletion Policy on Reasoning Time

Experiment Setup

This experiment collects 6 groups of data with increasing **Window Size** from 10 seconds to 60 seconds, and in each group, there are two values of **Reasoning Time** metric. One is collected with only **Deletion Policy** on, and the other one is collected with both **Deletion Policy** and **Property Filter** are on. In this experiment, the **longest period time** of the reasoning rule set is **10** seconds and the **Rule Complexity** is **7**.

Results

Figure 5.7 shows the evaluation results of this experiment.

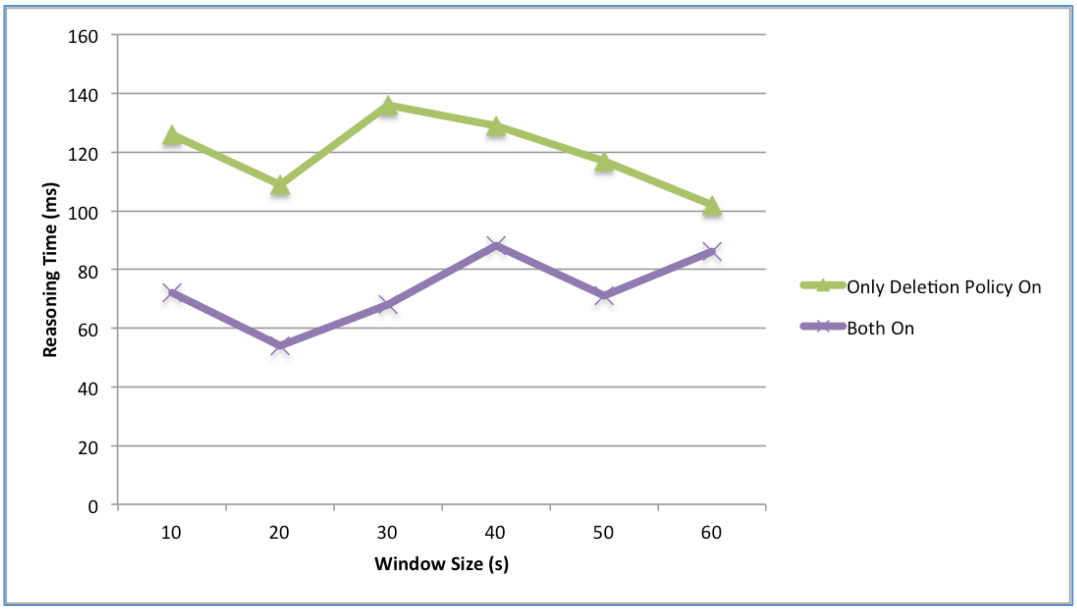


Figure 5.7 Property Filter *vs* Deletion Policy on Reasoning Time

Findings

According to the result, we could find that **Property Filter** could make further improvements even though the **Deletion Policy** is already on. The reason behind this is similar to previous analysis, which is, **Property Filter** is based on used properties, and **Deletion Policy** is based on longest time need to be reasoned.

5.2.2.4 Combined Results on Reasoning Time

Figure 5.8 shows the combined results of four different conditions, both filters are off, only **Property Filter** is on, only **Deletion Policy** is on, and both filters are on. We present all groups of data in one diagram to provide readers an overall view of how much improvement each filter could achieve, performance comparison between these two filters and different specifics each filter has.

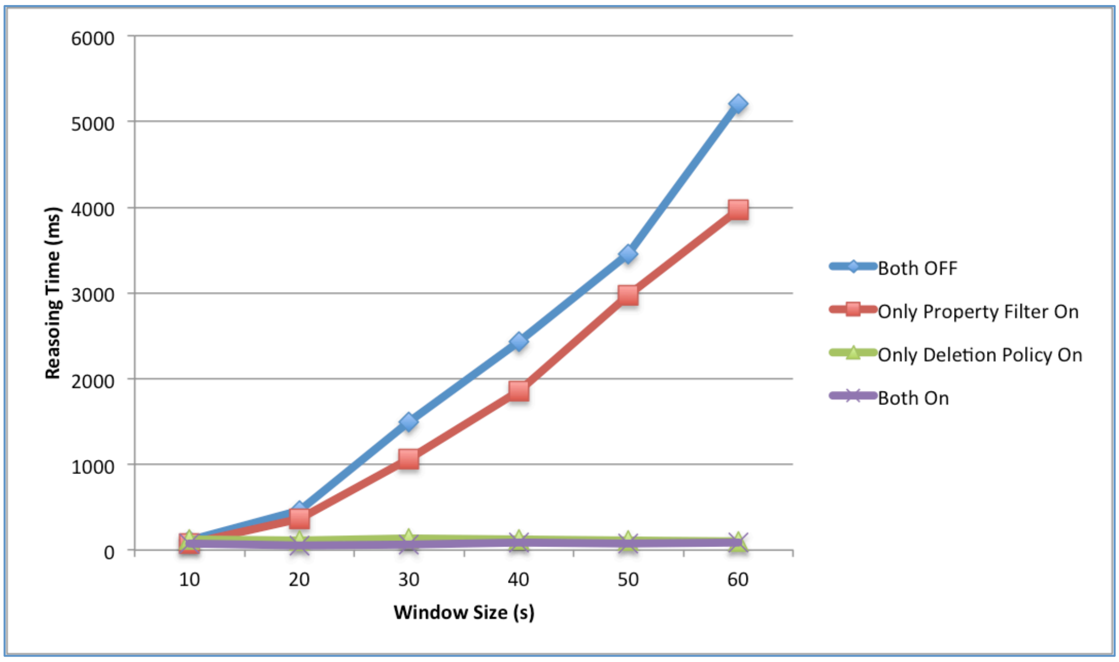


Figure 5.8 Combined Results on Reasoning Time

Findings Summary

In this series of experiments, the **Rule Complexity** of the rule set is **7** and **longest time period** is **10** seconds. According to Figure 5.8, we could summarize our findings as:

- **Property Filter** can reduce the reasoning time by around 25%.
- **Reasoning Speed** *decreases a lot* while window size or total number of triples increases.
- **Deletion Policy** can maintain the **Reasoning Time** under a much lower level even with increasing window size. In addition, the level of this flat line is still determined by the **longest time period** of rule set.
- **Property Filter** could still make further improvements on Reasoning Time even the **Deletion Policy** is already on.

5.2.3 Experiments about Memory Usage

5.2.3.1 Memory Usage *vs* Property Filter

Experiment Setup

This experiment collects 6 groups of data with increasing **Window Size** from 10 seconds to 60 seconds, and in each group, there are two values of **Memory Usage** metric. One is collected with no filter on, and the other one is collected with only **Property Filter** on. In this experiment, the **Rule Complexity** of the reasoning rule set is **7**, which is a medium level and longest time period is **10** seconds.

Results

Figure 5.9 shows the evaluation results of this experiment.

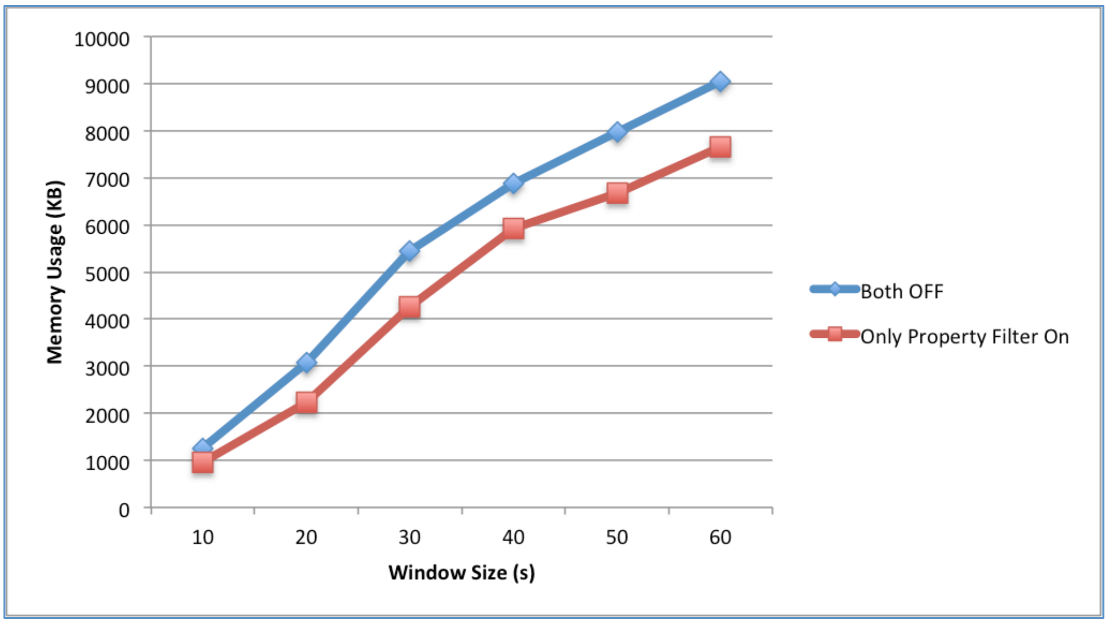


Figure 5.9 Memory Usage *vs* Property Filter

Findings

According to the result, we could find that Property Filter reduce the **Memory Usage** around 20% to 25%, which is still less than its affects on **Number of Triples**. The reason behind this is similar to previous analysis, which is because the **Memory Usage** also covers some other processes that are not only affected by the **Number of Triples**, including RETE network building [21], intermediate results caching and so on. Even though the inference is not as big as it made on **Number of Triples**, 20% to 25% improvement is significant enough for a filter mechanism. In addition, we could find that the **increase rate** of **Memory Usage** is getting smaller as the window size or total number of triples increases.

5.2.3.2 Memory Usage *vs* Deletion Policy

Experiment Setup

This experiment collects 6 groups of data with increasing **Window Size** from 10 seconds to 60 seconds, and in each group, there are two values of **Memory Usage** metric. One is collected with no filter on, and the other one is collected with only **Deletion Policy** on. In this experiment, the **longest period time** of the reasoning rule set is **10** seconds, which is a reasonable level and the **Rule Complexity** is **7**.

Results

Figure 5.10 shows the evaluation results of this experiment.

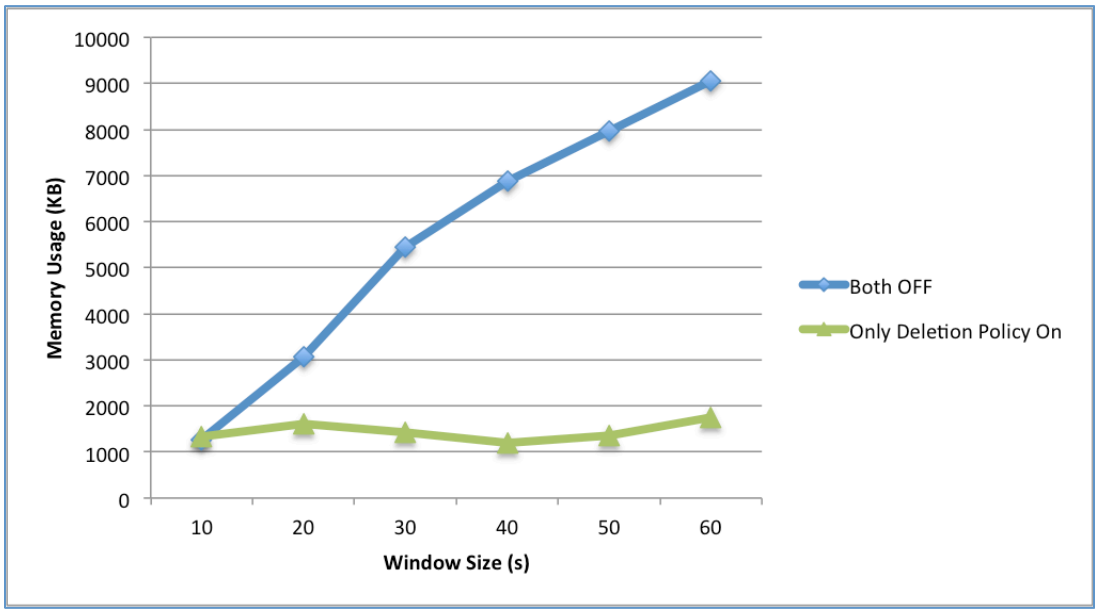


Figure 5.10 Memory Usage *vs* Deletion Policy

Findings

According to the result, we could find that **Deletion Policy** makes similar improvements on **Memory Usage** comparing to **Reasoning Time**. The reason behind is also similar. The **Deletion Policy** could maintain the total number of triples under a vey low level, which results in maintaining both the **Reasoning Time** and **Memory Usage** under a very low level as well. Again, the level of this flat line is determined by the **longest time period** of the rule set.

5.2.3.3 Property Filter *vs* Deletion Policy on Memory Usage

Experiment Setup

This experiment collects 6 groups of data with increasing **Window Size** from 10 seconds to 60 seconds, and in each group, there are two values of **Memory Usage** metric. One is collected with only **Deletion Policy** on, and the other one is collected with both **Deletion Policy** and **Property Filter** are on. In this experiment, the **longest period time** of the reasoning rule set is **10** seconds and the **Rule Complexity** is **7**.

Results

Figure 5.11 shows the evaluation results of this experiment.

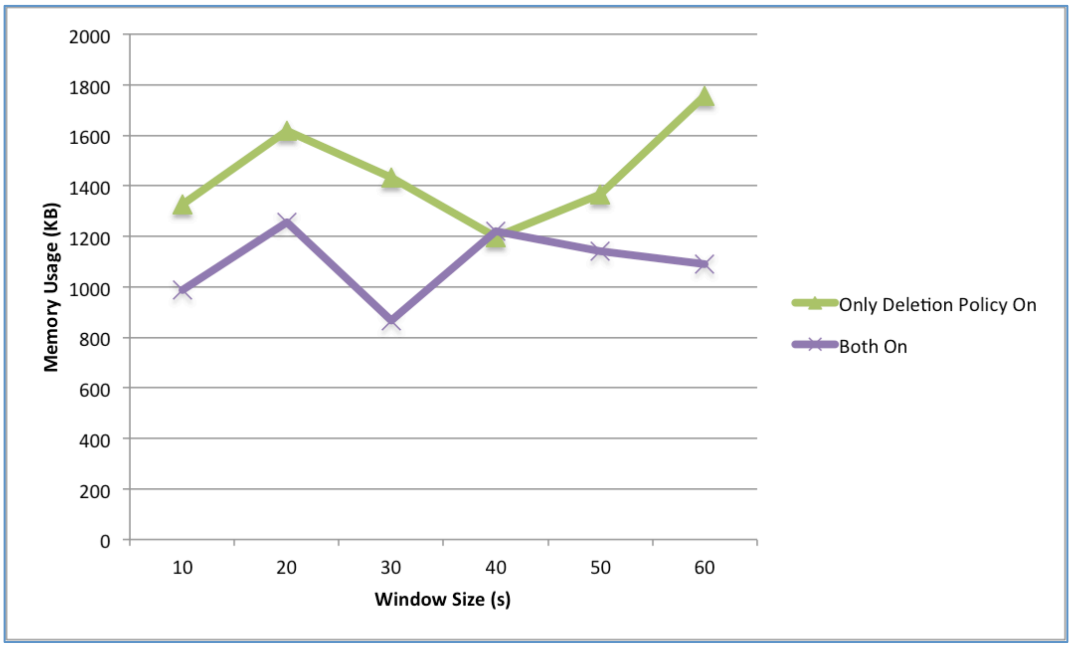


Figure 5.11 Property Filter *vs* Deletion Policy on Memory Usage

Findings

According to the result, we could find that **Property Filter** could still make further improvements on **Memory Usage** even though the **Deletion Policy** is already on. The reason behind this is similar to previous analysis, which is, **Property Filter** is based on used properties, and **Deletion Policy** is based on longest time need to be reasoned.

5.2.3.4 Combined Results on Reasoning Time

Figure 5.12 shows the combined results of four different conditions, both filters are off, only **Property Filter** is on, only **Deletion Policy** is on, and both filters are on. We present all groups of data in one diagram to provide readers an overall view of how much improvement each filter could achieve, performance comparison between these two filters and different specifics each filter has.

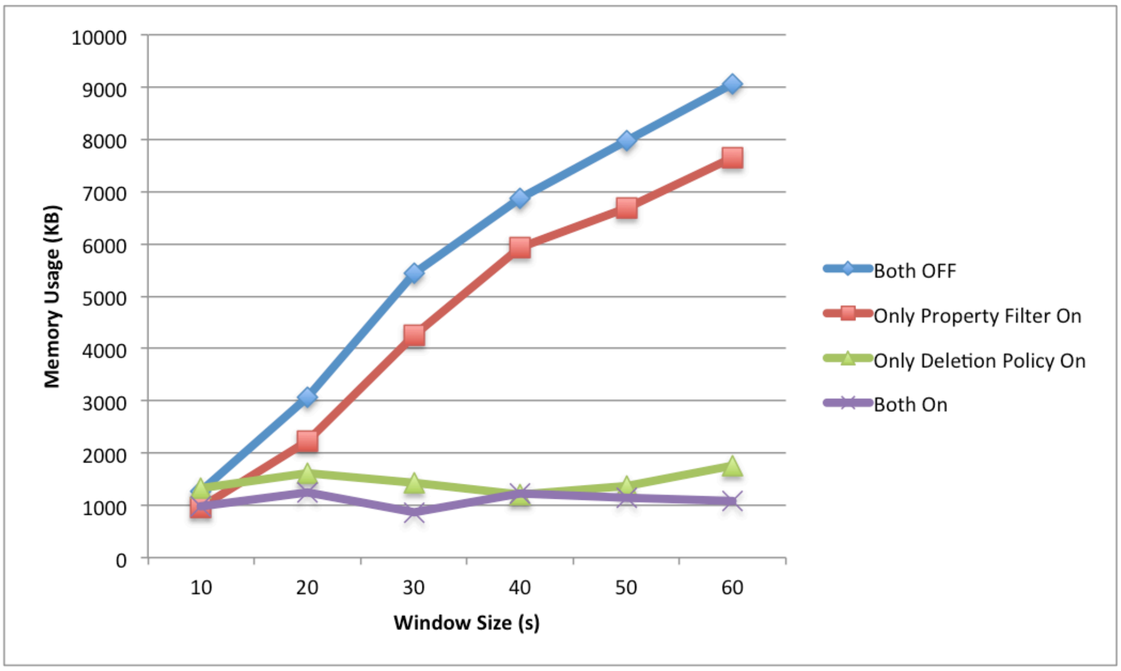


Figure 5.12 Combined Results on Memory Usage

Findings Summary

In this series of experiments, the **Rule Complexity** of the rule set is **7** and **longest time period** is **10** seconds. According to Figure 5.12, we could summarize our findings as:

- **Property Filter** can reduce the reasoning time by around 20% to 25%.

- The **increase rate** of **Memory Usage** is getting *smaller* as the window size or total number of triples increases.

- **Deletion Policy** can maintain the **Memory Usage** under a very low level even with increasing window size and this level is determined by the **longest time period** of rule set.

- **Property Filter** could still make further improvements on **Memory Usage** even the **Deletion Policy** is already on.

5.2.4 Experiments about Rule Complexity

This series of experiments evaluates how **Rule Complexity** inferences on the improvements that **Property Filter** made against, **Number of Triples**, **Reasoning Time** and **Memory Usage**.

5.2.4.1 Rule Complexity *vs* Property Filter on Number of Triples

Experiment Setup

This experiment collects 9 groups of data with increasing **Rule Complexity** from 1 to 17, and in each group, there are two values of **Number of Triples** metric. One is collected with no filter on, and the other one is collected with only **Property Filter** on. In this experiment, the **Window Size is** configured to **30** seconds.

Results

Figure 5.13 shows the evaluation results of this experiment.

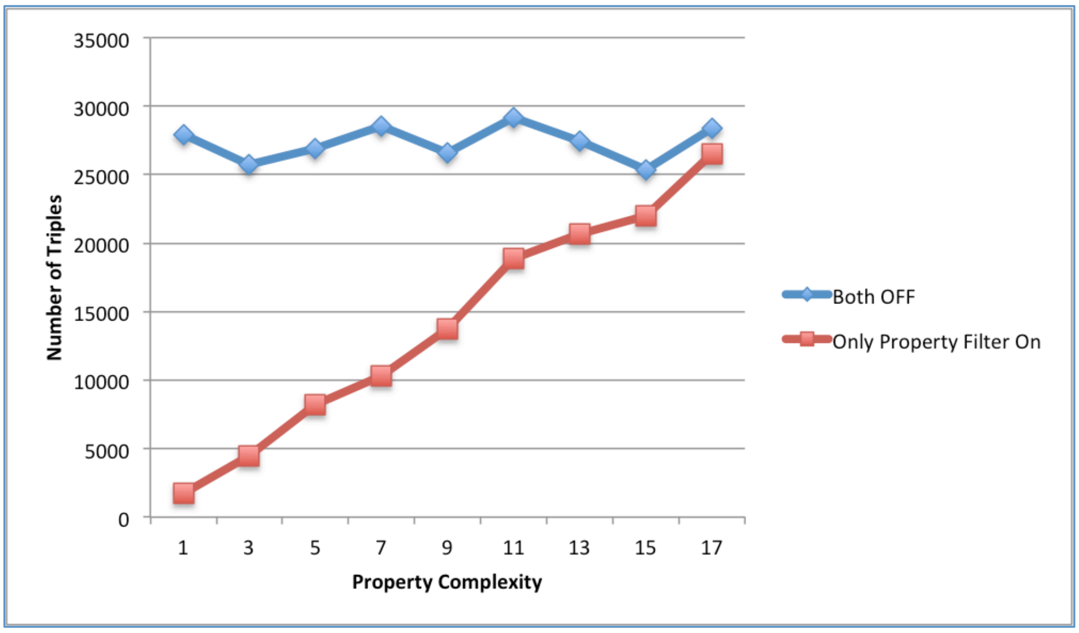


Figure 5.13 Rule Complexity *vs* Property Filter on Number of Triples

Findings

According to the result, it is very obvious that Rule Complexity has no inference on total number of triples. Once the **Property Filter** is on, the **Number of Triples** increases proportionally to Rule Complexity, because **Rule Complexity** has direct inference on **Number of Triples**, just like what we discussed in the previous analyses.

5.2.4.2 Rule Complexity *vs* Property Filter on Reasoning Time

Experiment Setup

This experiment collects 9 groups of data with increasing **Rule Complexity** from 1 to 17, and in each group, there are two values of **Reasoning Time** metric. One is collected with no filter on, and the other one is collected with only **Property Filter** on. In this experiment, the **Window Size is** configured to **30** seconds.

Results

Figure 5.14 shows the evaluation results of this experiment.

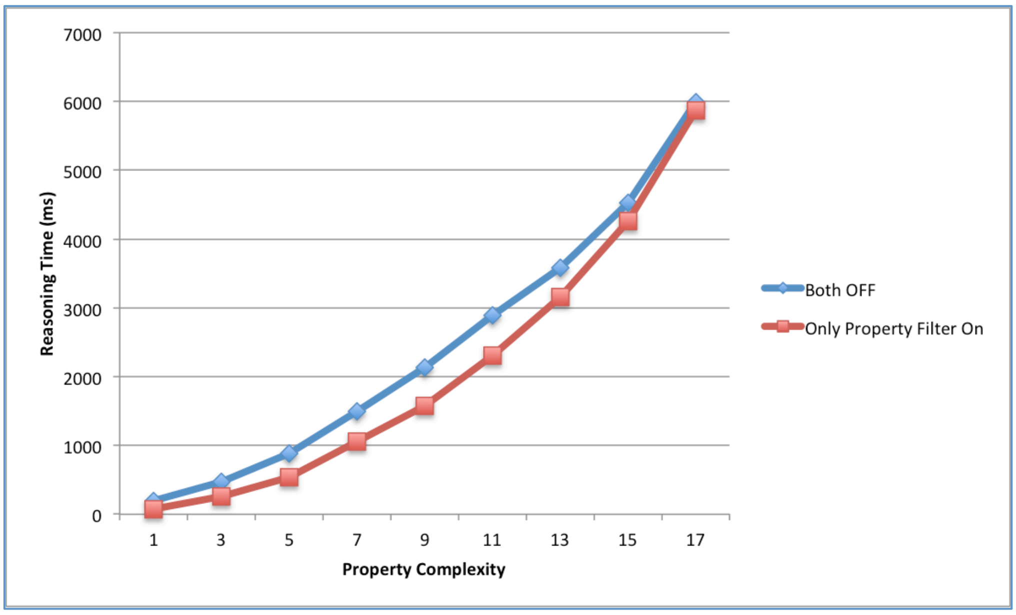


Figure 5.14 Rule Complexity *vs* Property Filter on Reasoning Time

Findings

According to the result, we could find that, when **Rule Complexity** is low, **Property Filter** could reduce the **Reasoning Time** up to 50%. However, as the Rule Complexity increases, fewer triples are removed by the **Property Filter** and more **Reasoning Time** is spent on building more complex RETE network or matching more complex patterns. Therefore, the **Reasoning Time** difference between these two lines are getting smaller, and finally, when the **Rule Complexity** is **17**, they reach the same point, where no triples are removed by the Property Filter.

In addition, this result diagram also provide strong support to our findings about **Reasoning Speed**, which *decreases a lot* as total number of triples increases.

5.2.4.3 Rule Complexity *vs* Property Filter on Memory Usage

Experiment Setup

This experiment collects 9 groups of data with increasing **Rule Complexity** from 1 to 17, and in each group, there are two values of **Memory Usage** metric. One is collected with no filter on, and the other one is collected with only **Property Filter** on. In this experiment, the **Window Size is** configured to **30** seconds.

Results

Figure 5.15 shows the evaluation results of this experiment.

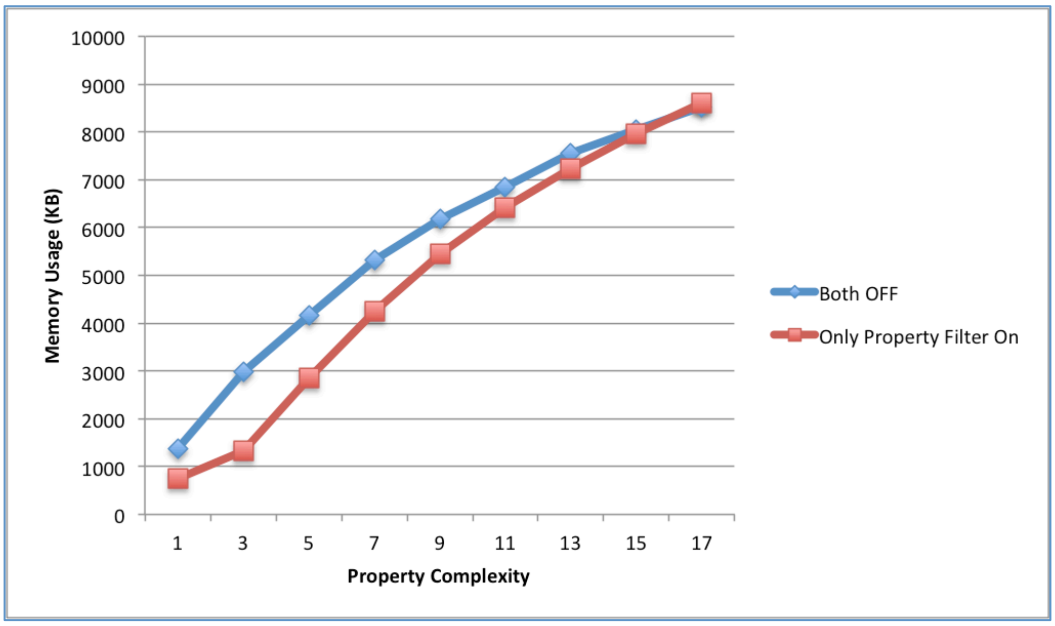


Figure 5.15 Rule Complexity *vs* Property Filter on Memory Usage

Findings

According to the result, we could find that, when **Rule Complexity** is low, **Property Filter** could reduce the **Memory Usage** up to 50%. However, as the Rule Complexity increases, fewer triples are removed by the **Property Filter** and more **Memory Usage** is spent on building more complex RETE network or caching more intermediate results. Therefore, the **Memory Usage** difference between these two lines are getting smaller, and finally, when the

Rule Complexity is **17**, they reach the same point, where no triples are removed by the Property Filter.

In addition, this result diagram also provide strong support to our findings about the **increase rate** of **Memory Usage**, which is getting *smaller* as the total number of triples increases.

5.2.5 Experiments about Reasoning Accuracy

Reasoning Accuracy is a very important metric to make our filter mechanisms being trusted and accepted by other researchers. No performance improvement is reasonable without the guarantee of **Reasoning Accuracy**.

Experiment Setup

It is not proper to analyze Reasoning Accuracy which is collected based on real time Twitter Stream, because even though we could make sure the window size stays the same across different experiment, we can't ensure the total number of triples and information contained in these triples stay the same. Therefore, we decide to conduct experiments against the same and static file with **30** seconds triples for 4 different configurations, including both filters are off, only **Property Filter** is on, only **Deletion Policy** is on, and both filters are on. For each experiment, we collect the total number of triples reasoned and total number of triples deduced.

In these experiments, the **Rule Complexity** of the rule set is 7. Moreover, in order to ensure **Temporal Functors** and **Deletion Policy** are still working and could produce the same result across different experiments, a static reference time is created to take place of dynamic system time, so that the only factors that could affect reasoning accuracy left are the filters.

Results

Table 5.1 shows the evaluation results of this experiment. Reasoning Accuracy is calculated based on Equation 5.1

Configuration	Triples Reasoned	Triples Deduced	Reasoning Accuracy
Both Filters Off	26878	1324	------
Only Property Filter On	13294	1324	100%
Only Deletion Policy On	8327	1324	100%
Both Filters On	3619	1324	100%

Table 5.1 Reasoning Accuracy *vs* Filters

Findings

According to the result, we can conclude that, no matter which filter mechanism is used, the Reasoning Accuracy is not affected.

5.3 Key Findings and Limitations

In this section, we summarize all the key findings concluded from previous experiments and also present the limitations of these filter mechanisms.

5.3.1 Key Findings

*Property Filter Performance (**Rule Complexity** of rule set is 7)*

- **Property Filter** can delete more than **50%** triples, which is proportional to **Rule Complexity**.
- **Property Filter** can reduce the **Reasoning Time** and **Memory Usage** by around **25%**.

- **Performance improvements** made by **Property Filter** are negatively related to **Rule Complexity**. More specifically, the improvement is getting smaller as **Rule Complexity** increases.

*Deletion Policy Performance (**Longest Time Period** of rule set is 10 seconds)*

- **Deletion Policy** can maintain the total **Number of Triples** under a very low level with increasing window size.

- **Deletion Policy** can maintain the **Reasoning Time** and **Memory Usage** under a much lower level with increasing window size, comparing to **Number of Triples**.

- These levels are determined by the **longest time period** of the rule set.

Reasoner Performance

- **Reasoning Speed** *decreases a lot* as **Window Size** or total **Number of Triples** increases.

- The **increase rate** of **Memory Usage** is getting smaller as **Window Size** or total **Number of Triples** increases.

-

5.3.2 Limitations

Property Filter Limitation

According to our key findings, the improvements made by **Property Filter** are related to **Rule Complexity**. Therefore, under some rare circumstances, the rule set could cover nearly all the 17 properties for each tweet, where the **Property Filter** could make very little contribution.

Property Filter Limitation

According to our key findings, the performance level that **Deletion Policy** maintains is determined by the longest time period of the rule set. Therefore, under very rare circumstances where the **longest time period** is equal to or greater than the **Window Size**, **Deletion Policy** could make no improvements at all.

5.4 Chapter Summary

In this chapter, we evaluated our system performance from different aspects, including number of triples reasoned, reasoning time consumed and memory usage to show the significant improvements we achieved from our novel filter mechanisms. In addition, reasoning accuracy is also examined to provide more confidence for other researchers about our system. Last but not the least, the limitation of our filter mechanisms are also presented which inspires further research and development.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Social media, like Twitter, is gradually becoming an important source of knowledge, which contains valuable information about a variety of industries. Semantic Web, believed as the next generation of World Wide Web, is very powerful on processing such high diversity data source.

In this dissertation, we present our approach to perform temporal reasoning on real time Twitter stream using Semantic Web Technologies. To achieve this, firstly, we design and implement our Twitter stream processor to capture real time Twitter data and convert it to RDF triples, which could be accepted by the semantic reasoner. In addition, we extend the previous semantic reasoner with temporal functors so that it can reason on time dimension data. Moreover, in order to process on Twitter's high-frequency and high-diversity data stream, we contribute two novel filter mechanisms to remove uninterested data based on different rule set characteristics. Last but not the least, we conduct a number of experiments to our system. According to the result we provide, we can conclude that the performance improvement achieved by these two filters is significant.

6.2 Future work

6.2.1 Design and Implement Incremental Reasoning

At the beginning of each reasoning process, a RETE network is built by the rule engine as the preparation for pattern matching algorithm. During the matching process, a huge number of matched result nodes are cached in memory and they will be released after the reasoner deduces the result.

As mentioned in previous chapters, our stream processor is a window-based approach. Therefore, each two adjacent windows have a certain period time of duplicate data, which is already reasoned in the first cycle. Incremental reasoning is a very powerful technology to solve this problem so that we could keep the intermediate matched result in memory and when a new window of data comes, instead of building the network again, only the expired data and its related matched result is removed from the RETE network and new data is added.

In theory, this should contribute very significant performance improvement. Therefore, it is very worthy of further researching and developing.

6.2.2 Extend to Reason on Other Data stream

Apart from Twitter data stream, there are a number of interesting and valuable data source that could be reason on. For example, Facebook. It is well worth extending our approach to perform temporal reasoning on Facebook stream and implement some new functors for specific scenarios. We are very interested to see how easy our approach could be extended, how well it could perform on other data stream and how it could be improved.

Abbreviations

RDF Resource Description Framework. 1, 5, 6

URI Uniform resource identifier. 6

RIF Rule Interchange Format. 7

W3C World Wide Web Consortium. 7

OO Object Oriented. 8

DL Description Logic. 9

CEP Complex Event Processing. 11

JDBC Java Database Connectivity. 29

IDE Integrated Development Environment. 29

Bibliography

[1] J. R. Hobbs and F. Pan, “An ontology of time for the semantic web,” *ACM Transactions on Asian Language Information Processing*, vol. 3, no. 1, pp. 66–85, Mar. 2004.

[2] C. Bizer, T. Heath, K. Idehen, and T. Berners-Lee, “Linked data on the web (LDOW2008),” *Proceeding of the 17th international conference on World Wide Web - WWW ’08*, pp. 1265–1266, 2008.

[3] N. Shadbolt, T. Berners-Lee, and W. Hall, “The Semantic Web Revisited,” *IEEE Intelligent Systems*, vol. 21, no. 3, pp. 96–101, May 2006.

[4] J. R. Hobbs and F. Pan, “An ontology of time for the semantic web,” *ACM Transactions on Asian Language Information Processing*, vol. 3, no. 1, pp. 66–85, Mar. 2004.

[5] C. Gutierrez, C. Hurtado, and A. Vaisman, “Introducing Time into RDF,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 2, pp. 207–218, Feb. 2007.

[6] J. Bian, U. Topaloglu, and F. Yu, “Towards large-scale twitter mining for drug-related adverse events,” *Proceedings of the 2012 international workshop on Smart health and wellbeing*, pp. 25–32, 2012.

[7] M. G. Barbieri, Davide, Daniele Braga, Stefano Ceri, Emanuele Della Valle,, “Stream reasoning: Where we got so far,” *Proceedings of the 4th workshop on new forms of reasoning for the Semantic Web: Scalable & dynamic*, pp. 1–7, 2010.

[8] D. Anicic, S. Rudolph, P. Fodor, and N. Stojanovic, “Stream reasoning and complex event processing in ETALIS,” *Semantic Web, IOS Press*, vol. 1, pp. 1–5, 2012.

[9] M. Kifer, H. Boley, “RIF Overview,” *W3C Working Group, 2013.*

[10] J. Carroll, I. Dickinson, and C. Dollin, “Jena: implementing the semantic web recommendations,” *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pp. 74–83, 2004.

[11] G. Meditskos and N. Bassiliades, “A Rule-Based Object-Oriented OWL Reasoner,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 3, pp. 397–410, 2008.

[12] T. Kim, I. Park, S. J. Hyun, and D. Lee, “MiRE4OWL: Mobile Rule Engine for OWL,” *2010 IEEE 34th Annual Computer Software and Applications Conference Workshops*, pp. 317–322, Jul. 2010.

[13] S. Ali and S. Kiefer, “μOR–A Micro OWL DL Reasoner for Ambient Intelligent Devices,” *Advances in Grid and Pervasive Computing*, pp. 305–316, 2009.

[14] W. Tai, J. Keeney, and D. O’Sullivan, “COROR: a composable rule-entailment owl reasoner for resource-constrained devices,” *in Proceedings of The 5th International Symposium on Rules: Research Based and Industry Focused (RuleML’11)*, pp. 212–226, 2011.

[15] W. Tai, J. Keeney, and D. O’Sullivan, “Resource-Constrained Reasoning Using a Reasoner Composition Approach,” *semantic-web-journal.net*, 2013.

[16] E. Della Valle, S. Ceri, P. Milano, F. Van Harmelen, and V. U. Amsterdam, “It ’ s a Streaming World ! Reasoning upon Rapidly Changing Information,” *Intelligent Systems, IEEE*, vol. 24, no. 6, pp. 83–89, 2009.

[17] D. Barbieri, D. Braga, and S. Ceri, "Deductive and Inductive Stream Reasoning for Semantic Social Media Analytics," *Intelligent Systems, IEEE*, vol. 25, no. 6, pp. 32–41, 2010.

[18] J. Keeney, C. Stevens, and D. O'Sullivan, "Extending a knowledge-based network to support temporal event reasoning," *2010 IEEE Network Operations and Management Symposium - NOMS 2010*, pp. 631–638, 2010.

[19] C. Gutierrez, C. Hurtado, and A. Vaisman, "Introducing Time into RDF," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 2, pp. 207–218, Feb. 2007.

[20] S. Batsakis, "SOWL : Spatio-temporal Representation , Reasoning and Querying over the Semantic Web Categories and Subject Descriptors," *Proceedings of the 6th International Conference on Semantic Systems*, pp. 1–9, 2010.

[21] K. Walzer, T. Breddin, and M. Groch, "Relative temporal constraints in the Rete algorithm for complex event detection," *Proceedings of the second international conference on Distributed event-based systems - DEBS '08*, p. 147, 2008.

[22] F. Heintz, J. Kvarnström, and P. Doherty, "Stream reasoning in dyknow: A knowledge processing middleware system," *1st Int'l Workshop Stream Reasoning*, pp. 83–89, 2009.

[23] J. C. Augusto, "Temporal reasoning for decision support in medicine.," *Artificial intelligence in medicine*, vol. 33, no. 1, pp. 1–24, Jan. 2005.