CEP-ML: META-LANGUAGE TO SUPPORT INTEROPERABILITY BETWEEN HETEROGENEOUS COMPLEX EVENT PROCESSING SYSTEMS

W.D. Amila Iroshani Paranawithana

(158231T)

Degree of Master of Science

Department of Computer Science & Engineering

University of Moratuwa
Sri Lanka

May 2019

CEP-ML: META-LANGUAGE TO SUPPORT INTEROPERABILITY BETWEEN HETEROGENEOUS COMPLEX EVENT PROCESSING SYSTEMS

W.D. Amila Iroshani Paranawithana

(158231T)

Thesis submitted in partial fulfilment of the requirements for the degree Master of Science in Computer Science and Engineering

Department of Computer Science & Engineering

University of Moratuwa
Sri Lanka

May 2019

DECLARATION

I declare that this is my own work and this thesis does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to the University of Moratuwa the non-exclusive right to reproduce and distribute my thesis, in whole or in part in print, electronic or another medium. I retain the right to use this content in whole or part in future works (such as articles or books).

....

W.D. Amila Iroshani Paranawithana Date

The above candidate has carried out research for the Master of Science thesis under my supervision.

....

Dr. Surangika Ranathunga Date

ABSTRACT

Distributed complex event processing systems give many benefits over centralized systems mainly in terms of scalability and extendibility. There are many types of CEP engines with different characteristics and query languages specialized to each domain. When it comes to deploying these distributed CEP systems in an industrial context, supporting interoperability between these heterogeneous event processing systems has become a major problem.

Not having a generally accepted definition language is a prime problem when integrating different CEP engines to achieve one goal in a distributed environment. There have been introduced new systems and languages to be operated efficiently in a distributed environment but, they have not addressed the problem of not having a generally accepted language when communicating between different CEP engines. There has been little quantitative analysis done on developing a meta-language and a language conversion parser. The absence of a language parser to convert between any available meta-language and other existing CEP languages is another noticeable shortage when migrating between different CEP systems.

This research presents a generally accepted definition meta-language for complex event processing to support interoperability between CEP systems along with a language parser to convert between this meta-language and existing languages. It acts as an intermediate language format in language conversion. The meta-language supports the main common language functions to reach the industrial level. CEP ML language parser supports three popular languages SiddhiQL, EPL and Stream that have dominated the field for years. Further, we have developed a web-based try-out tool which users can easily use to convert between these languages.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor Dr. Surangika Ranathunga of the Computer Science Department at the University of Moratuwa for her continuous support for completing my research and thesis. Her motivation, patience and continues support always helped to follow the right path and resolve problems I faced during this journey.

I would like to thank all the staff from the Department of Computer Science and Engineering, University of Moratuwa, for their kindness they expressed on all occasions.

Finally, I must convey my sincere gratitude to my parents, husband and friends for their unfading support and continual motivation throughout this journey of my masters. This achievement would not have been realizable without all of them. Thank you.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Description
API	Application Programming Interface
BPM	Business Process Management
CEP	Complex Event Processing
CPU	Central Processing Unit
CQL	Continuous Query Language
EPL	Event Processing Language
JAXB	Java Architecture for XML Binding
REST	Representational State Transfer
SQL	Structured Query Language
SiddhiQL	Siddhi Query Language
UI	User Interface
WAR	Web application Archive
XML	eXtensible Markup Language

1. INTRODUCTION

1.1. Overview

Timely processing and analysing streams of data to detect situations and the ability to respond quickly is very advantageous in competitive business environments. Some examples are, monitoring continuous streams of transaction data to detect fraud patterns in financial applications, analysing sequences of measurements generated from sensor networks to give quick responses for physical changes in environment pollution levels, and even to detect natural disasters, analysing vast amount of data generated from social media to interpret people's opinions and in many more domains such as healthcare, telecommunication and sports. Traditionally this was done with conventional database systems where data is stored and indexed and queried later [1].

Complex Event Processing (CEP) is a widely used technology to analyse event data streams to detect patterns in near real-time based on user-defined complex queries [2]. Using complex event processing systems, we can detect complex events matching the high-level queries defined by users and respond in near real time. These predefined CEP rules are written in a specifically designed query language similar to SQL.

This was originally done with centralized CEP systems where all the event streams were processed by a single system. But with the rapid increase of event sources centralized CEP systems gave many disadvantages such as single point of failure, communication overhead that degrades the performance, increasing the response time, and causing undesirable processing cost.

To address these issues with centralized CEP systems, the requirement of distributed CEP systems emerged. Distributed CEP systems give many advantages in terms of scalability, extensibility and reliability. It provides required computational power to analyse event streams generated in varying rates with its auto scaling capability and ensure the reliability and avoids single point of failure. But in current distributed CEP systems there are issues in functionality, expressiveness, reusability of present

knowledge, user-friendly interfaces, flexibility and interoperability. Figure 1.0 shows the overall idea of how data from different sources are sensed and streamed into a distributed CEP system to process parallelly in a distributed manner to give meaningful results to the users in near real time.

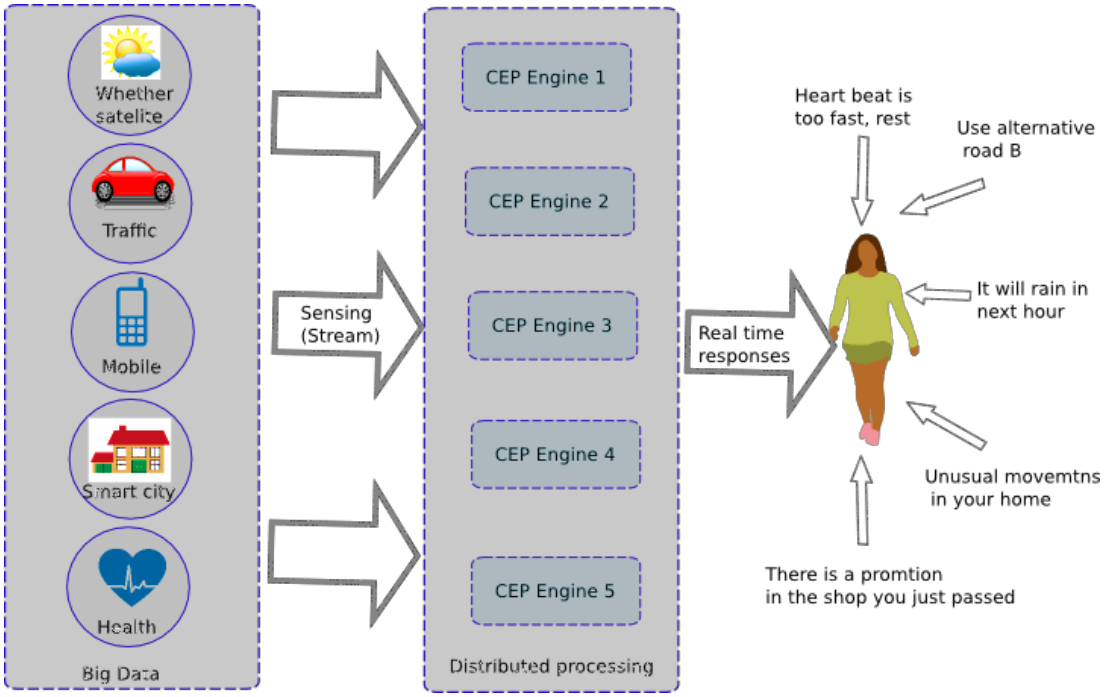


Figure 1.0 Distributed Complex Event Processing

1.2. Problem and Motivation

In distributed CEP systems, different types of CEP products are used due to many reasons. Each CEP system is specialized in different domains such as business process management (BPM), financial services, cyber-physical systems, IoT, health, aerospace industry, transportation and each task may have different complexities requiring different processing capabilities and functionalities provided by different CEP engines. There are many types of CEP engines that we can mainly divide into two categories as aggregation-oriented CEP, which are focused on executing on-line algorithms and detection-oriented CEP, which are focused on detecting event patterns [1]. When communicating with cross platforms another issue needs to be

contemplated is the licensing issue. It is not possible for one company to accommodate licenses for several CEP products. Also, due to restrictions that come with different company policies, it has become difficult to easily interconnect with each of these systems.

In the business scenarios intercommunication between these different CEP systems is an imperative requirement. For an example, to improve the sales in a business by sending promotions at the right time and right place to the most relevant customers, a business will require to consolidate and share the analysed data generated by different CEP engines installed at different domains such as customer sales tracking systems, customer location/activity tracking systems and environmental data analysis systems. Some of these tracking systems might be operated by third parties, who may have to abide by their licensing issues, or who may already have their preferred CEP engine.

With regard to integration of these heterogeneous CEP technologies, there are many problems. One of the fundamental problems is that the unavailability of a generally accepted query processing language for CEP [3] like SQL for database queries. Different CEP languages have different language formats and support different types of functionalities.

According to the best of our knowledge, there is only one common rule language introduced for CEP languages so far, which is RuleML [4]. But, RuleML is not up to the standard of industrial usage as it does not support some of the fundamental language functions that are common in existing query languages. Also, it does not have a language parser to convert RuleML queries to other existing popular CEP languages which is utterly important in distributed CEP systems. There are also some frameworks available such as DHEP [1] to coordinate between heterogeneous CEP engines but has not focused on resolving the language translation issue.

New CEP systems have been introduced with new query languages to detect event patterns in a parallelized or distributed manner. Poul, Migliavacca and Pietzuch [5] in their research introduce a new CEP system with a SQL like high level query language which facilitates rewriting of expressions into optimized, equivalent queries based on the resource consumption of event automata and executing those queries in the

distributed system. Cayuga [6, 7] CEP system also introduces a new query language with six operators and an automated system for event pattern detection in a distributed system. DistCED system [8] also proposes a new CEP language and a automata distribution system which can address the problem of incorrect detection due to network latency.

Even though the aforementioned solutions improve the performance and correctness of distributed event processing, they have not captured the issue of executing queries in different types of CEP engines which is the prime concern in this research. Hence, these systems do not have a general language or a language parser to coordinate with heterogeneous CEP systems that are already deployed or need to be deployed due to various reasons as mentioned above. CEP ML and its language parser focus on addressing that issue.

Another issue of not having a general CEP query language is when migrating from one CEP engine to another, having to rewrite all existing queries to convert to new CEP engine query type. Due to the unavailability of a language parser, this query conversion has to be done manually or custom language parsers have to be implemented, which is expensive and time consuming.

1.3. Objectives

Objectives of this research are twofold.

1. Introducing a generally accepted meta-language for CEP queries.

Under this existing CEP query languages will be analysed and categorized in order to identify the common functionalities and propose a common language syntax for facilitating those functionalities.

This language will be used as an intermediate language in CEP query conversion. When communicating between different systems in a distributed environment and when migrating to a new CEP system, CEP ML can be used as an intermediate language format to convert queries from one language format to another.

2. Implementing a language parser that can convert from one language to another through the meta-language. In this research, the following most common CEP languages will be taken into consideration.

    a. SiddhiQL of Siddhi CEP

    b. EPL of ESPER

    c. CQL of STREAM

1.4. Contributions

This project makes the following contributions:

- Generally accepted meta-language CEP ML for CEP queries.
- Language parser to convert a given existing language to CEP ML and vice versa. This supports 3 existing languages.
- Web based language conversion try-out tool

1.5. Organization of the Thesis

The rest of the thesis is organized as follows. Chapter two reviews related work in elaborating challenges in interoperability between heterogeneous CEP engines focussing more on lack of common meta-language. It further reviews on CEP query language categorization in order to identify the basic common rules and capabilities of existing languages. Finally, it discusses about existing solutions available and their lacks and limitations.

The third chapter is concerned with the methodology used for this study. It explains the approach used in designing the new meta-language and the language parser.

Chapter four evaluates the work presented in the thesis. It discusses on how the presenting solution addresses the prime objective of this research and its unique features.

Finally, in chapter five the conclusion gives a brief summary and critique of the findings and it also mentions the areas identified for further research and improvements for this research.

2. LITERATURE SURVEY

2.1. Overview

This chapter focuses on the challenges faced when interoperating between heterogeneous CEP engines, the significance of having a general query language and studies on existing CEP query languages. Under that, this chapter presents a categorization of query languages based on their characteristics and further on the extensibility of different query types of these languages. As this research is scoped on developing the language parser for three major CEP languages SiddhiQL, EPL and CQL a basic introduction on these languages and a comparison between their language function capabilities will be discussed here. It also discusses existing solutions such as DHEP system and RuleML language and their deficiencies in addressing this problem.

2.2. Complex Event Processing engines

Today there is so much data that get generated in every second from various sources such as sensors, web activities, transactions, social networks etc and from different applications such as e-Science use-cases, business applications, financial trading applications, operational analytics applications and business activity monitoring applications. To make use of these data in an effective and utilitarian manner data processing has become a vital necessity. Complex Event Processing main and rapidly emerging technology solution that is widely used for real-time data processing. A Complex Event Processor identify relevant relationships and patterns from different streams of events and confine it into a composite event in order to send to other relevant components [9].

2.3. Challenges in interoperability between heterogeneous CEP engines

Even though the distributed CEP systems grant many advantages, in the real industrial environments interoperability between these distributed event processing systems is a complex problem that has not been solved into an acceptable level.

Language translation is an unavoidable fact in interoperability and direct transition between one syntax to another is fairly complex because those are not originally designed to that. Also, this language transition introduces a large overhead since all the nodes need to be aware of all possible counterparts [10].

Apart from query language translation, there are many more challenges when considering interoperability between heterogeneous systems. One is the security issues since correlation rules may contain confidential business process information which domain owners are reluctant to share. Different CEP engines may be running in different networks and in order to work collaboratively between these systems events and rules have to be exchangeable, so the communication is one challenge.

The DHEP (Distributed Heterogeneous Event Processing) system [1] introduces a framework that can embed centralized CEP engines to create a distributed processing system which manages communication between the nodes and distribution of rules. As per the best of our knowledge, DHEP framework is the only solution so far which address the challenge of interoperation between heterogeneous CEP engines to some extent even though it has some issues. They have introduced a meta-language that allows to design and manage events, rules and context information within the distributed system. The rule management component of the DHEP offers interfaces to move rules to other nodes and deploy them in local processing engines and provide rule translators to support each CEP engine type. DHEP provide a whole framework with set of functionalities such as Event Bus which distribute the data, Decoder/Encoder component, Routing component which route the incoming events according to a routing table and Wrapper which is responsible for the integration of the different engines and all these require a considerable processing power which may not be available in some practical environments. However, this meta-language supports limited correlation operators such as basic SEQ, ALL, OR, and NOT. Also,

some rules may not be placed on some nodes, because its restrictions do not match the nodes attributes and the resource usage of these rules are high.

2.4. CEP query language categorization

To create a meta-language for CEP query languages a comprehensive study on the existing languages is essential. A survey by Lai-Ham [10] has identified types of complex events and analysed the expressibility of these complex events with existing, selected CEP query languages by considering real life sample scenarios. CEP languages can be mainly categorized into three as below.

a. Data stream query languages

Data stream query languages is to query streams of data/events which mainly use a relational query language like SQL. In this method, a set of data in a stream at a time instance is converted into a relation and queries are executed and results are converted back to a stream as shown in Figure 2.1. The main three types of operators of data stream query languages are relation-to-stream operators, stream-to-relation operators and relation-to-relation operators.

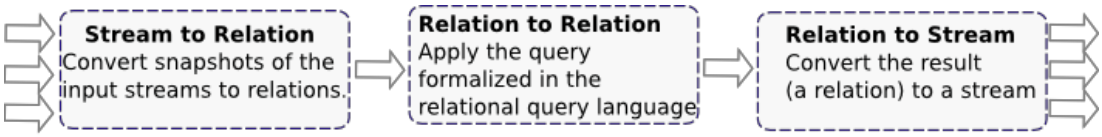


Figure 2.1 Data stream query languages operation pattern [11]

Also retrieving select events can be done by applying a window such as tuple windows to retrieve only the last n events, and time windows to retrieve only the events that entered the stream in the last n time units. Esper, STREAM, Siddhi are sample CEP engines use this type of CEP languages. Coral8 [cor], Avaya Event Processor [ava], BEA (Oracle) Complex Event Processing [bea], and StreamBase [strb] are commercial CEP engines of this kind [11].

a. Composition-operator-based languages

Composition-operator-based languages can be defined as the composition of complex event queries using small, simpler event queries. The supporting operators of these languages which determine expressivity of these languages are conjunction, disjunction, sequences, negation, counting, and applicability of constraints. AMiT (IBM Active Middleware Technology) is a sample for this type [11].

b. Production rule languages

Production languages are used to express production rules that are then deployed in a production rule engine. Those rule languages do not operate on streams, but on data structures called working memories: mutable sets of objects capable of carrying data, called facts. Production rules consist of a condition which checks the existing working memory and action which change the working memory by adding, removing, or altering facts. Drools which is also known as JBoss rules is a sample for this type [11].

2.4.1. Expressibility of different query types by CEP query languages

Table 2.2 lists different types of queries that are supported by query languages. Table 2.3 shows the expressibility of each query type by languages with the symbols defined in Table 2.1 [11].

Table 2.1 Symbolic meanings

Symbol	Meaning
+	Fully expressible using desired features
⊕	Partially expressible using desired features (desired features insufficiently present)
⊖	Fully/partially expressible using other features (desired features not present, or requires not generally applicable “tricks” such as additional streams or low-level coding to work, or insufficient documentation)
–	Not expressible

Table 2.2 Different query types [11]

Q1	Disjunction
Q2	Negation, time windows
Q3	Conjunction, data extraction
Q4	Using external data sources
Q5	Tuple windows, aggregation by group
Q6	Counting
Q7	Aggregation
Q8	Event instance selection
Q9	Sequences
Q10	Event instance consumption

Table 2.3 Expressibility of each query by language [11]

Language	Group	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
STREAM	DS	+	⊖	+	⊖	+	+	+	⊖	⊖	⊖
Borealis	DS	+	–	+	⊖	+	+	+	–	⊖	–
AMiT	CO	+	⊖	+	–	–	+	–	+	+	+
ruleCore	CO	+	⊕	+	–	–	⊕	–	⊖	⊕	⊕
SASE+	CO	+	⊖	+	–	+	+	+	–	+	⊖
Esper	DS, CO	+	+	+	+	+	+	+	+	+	⊖
Cayuga	DS, CO	+	–	+	–	+	+	+	–	–	⊖

Drools	PR	+	⊖	+	⊖	⊖	+	+	+	+	+
XChange EQ	Other	+	+	+	⊕	⊖	+	+	–	⊕	–
SiddhiQL	DS	+	+	+	⊕	+	+	+	⊕	⊕	⊕

Considering the results of the language categorization and the popularity of the CEP engines used in the industry this project considers three languages SiddhiQL of Siddhi, EPL of ESPER and STREAM languages to analyse further and the parser is implemented for these languages. Since these three languages belong to data stream query language category, they have a SQL like a query pattern and has similar expressibility.

2.4.2. EPL in ESPER

Esper is an open source CEP engine for event streams processing, analysis and event correlation. Its query language, Event Processing Language (EPL) is a well-established rich language which can express filtering, aggregation, joins and sliding windows of multiple event series. It also includes pattern semantics to express complex temporal causality among events (followed-by relationship) [12].

EPL is a SQL a like language which has the usual features of a data stream language and has also inspired by composition- operator-based languages. It offers SELECT, FROM, WHERE, GROUP BY, HAVING and ORDER BY clauses [13].

2.4.3. SiddhiQL in Siddhi

WSO2 Complex Event Processor is a open source Complex Event Processing server which can handle up to throughput of nearly 100K+ events per second on single-server commodity hardware.

SiddhiQL [14] is a SQL like a language. Main language constructs of siddhiQL are event stream definitions, event table definitions, partitions and queries. SiddhiQL supports many functions such as FILTER, WINDOW, GROUP BY, SEQUENCES, JOIN, HAVING, output rate limiting... etc.

2.4.4. CQL in STREAM

STREAM (STanford stREam datA Manager) was a result of a research project done at Stanford University. STREAM CEP engine has its own Event Query Language called Continuous Query Language (CQL) which the syntax is very similar to SQL. STREAM language has strongly influenced EPL in ESPER and CCL in Coral8 [15]. The main operations supported by CQL are WHERE, GROUP_BY and HAVING. Also, it supports time windows and tuple windows for converting streams-to-relations. CQL provide Istream(insert streams), Rstream(relation streams) and Dstream(delete streams) as relation-to-stream operators [17, 18].

2.4.5. Comparison on Functions of EPL, SiddhiQL and CQL

Following Table 2.4 compare the main functions provided by 3 query languages which are considered in this paper EPL, SiddhiQL and CQL. (This list does not include all the functions of each language and considers only main functions)

Since EPL, SiddhiQL and CQL all belongs to Data Stream Language type, they have similar operators with similar syntax. Filtering from each stream is only supported in EPL and SiddhiQL wherein CQL this is done in ‘where’ condition. Inserting the results to another stream is supported only in EPL and SiddhiQL. In stream joining, EPL supports advanced joins such as LEFT, RIGHT, OUTTER joins. Aggregate functions supported by all 3 languages are almost similar in functions and syntax. Window filtering is similar in EPL and SiddhiQL but CQL does not support advanced window functions.

Table 2.4 Comparison of functions of query languages

Query Function	**EPL**	**SiddhiQL**	**CQL**
Projection	SELECT	SELECT	SELECT
Filter	FROM <> [<filter condition>]	FROM <>[<filter condition>]	
Window	.win:time([time peiod])	#window.<window> (<parameters>)	[Range <period>] / [Now]
Output event categories		Current/ expired/all events	
Aggregate	function(variable) sum, avg, median, stddev , avedev, count	function(variable) sum, avg, max, min, count, stddev	function(variable) Average, max, min
Group By	group by	group by	group by
Having	having	having	having
Output Rate Limiting	output [all | first | last | snapshot]	output ({<output-type>} every (<time interval>|<event interval> events) | snapshot every <time interval>)	output
Joins	, (comma)(supports Outer, left, right joins)	join	, (comma)
Remove duplicates	SELECT distinct		Distinct
Search condition	WHERE <condition>	ON	WHERE <condition>
Limit row count	limit <row count>		
Inserting events into table	Insert into / insert <> into <>	Insert into	
Sub queries	supported		
Deleting events from event table		delete <table name> on	
Update events in event table		Update <table name> on	

2.5. RuleML

To interchange web rules in XML format, RuleML [4] (Rule Markup Language) has been designed where it supports various rule languages. It allows for the exchange of rules between many heterogeneous systems. For example, distributed software components on the Web, heterogeneous client-server systems found within large corporations and complex event processing systems.

2.5.1. Reaction RuleML for CEP

In RuleML there are rules that execute over real time flowing data and expecting response in near real time, which are called reaction rules. Reaction rules is mainly based on ontologies of complex events. It consists of a semantic interchange format and standardized rule markup language. There is an extension of Reaction RuleML for Complex Event Processing (CEP) which detect complex events and reaction in near real time. This language includes different types of terms, formulas and performatives. Figure 2.2 shows the taxonomy of RuleML.

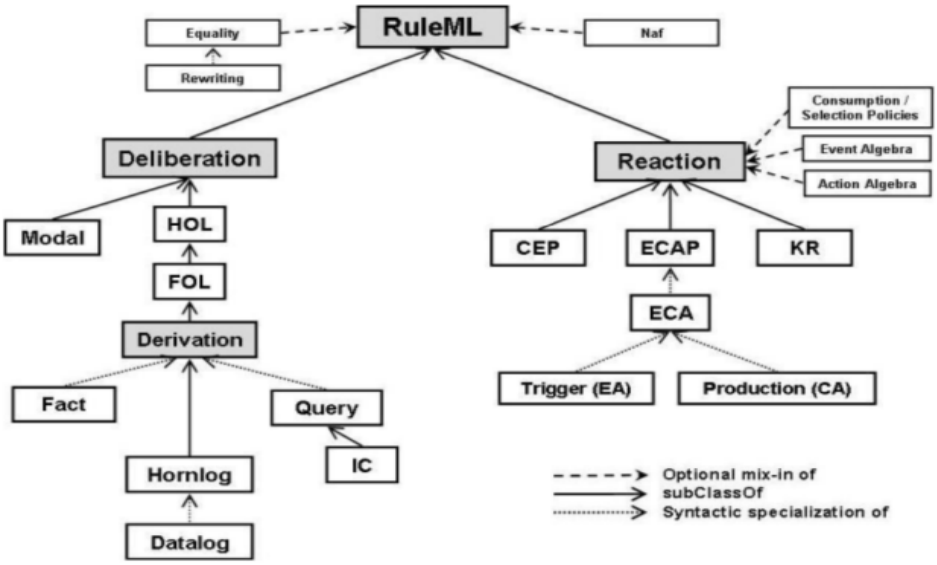


Figure 2.2 Taxonomy of RuleML rule

CEP RuleML supports xml tags <on> <Receive> which defines on which event receive to be activated, <do> - the query to execute on incoming data, <if> - to check some conditions are supported. Figure 2.3 is a sample reaction Reaction RuleML query.

```
<Rule style="active">
    <on><Receive> receival of event from 'A' </Receive></on>
    <do><Send> query 'B' for regular products in a new sub-
conversation </Send></do>
    <on><Receive> receive results from sub conversation with B
</Receive></on>
    <if> check some conditions </if>
    <do><Send> send results received from 'B' back to 'A'
</Send></do>
</Rule>
```

Figure 2.3 CEP RuleML sample

2.5.2. RuleML limitations

RuleML was originally created to support web rules in general and later came to support CEP rules with introducing reaction-RuleML. Hence, it is not designed to fully support the fundamental CEP query language features and it is still have not improved up to the standard of industrial usage even in their latest reaction RuleML specification 1.02 [16]. However, when considering using RuleML to define CEP rules, it has some disadvantages. One is XML is a heavy language with a large size of metadata and also its' syntax is too verbose. Since RuleML has not currently implemented parsers to convert CEP RuleML to other industrial using CEP languages users need to create parsers to their required language. Currently, CEP RuleML does not support most common query functions such as windows, aggregate functions, sequences, grouping etc. It is a bit hard to extend it to support these functions with the limitations in the hierarchical model in XML.

2.6. XML for query language designs

When considering the existing CEP query languages in the industry, there seem to be many types of formats used. For example, relational query language like SQL, object based, and XML based.

XML is one of the popular formats used for constructing languages. Novak and Marek [19] in their research describes how convenient to use XML to create a new domain specific language. This paper [20] describes what are the advantages of using XML as language format in constructing a language

- Easy to understand and manipulate - well-structured with a fixed schema.
- Extensible - good for evolving languages
- Widely supported - tools are already available for reading/parsing xml
- Human Readable - even non-technical people can understand

Also, there are many tools already available to parse XML tree structure and to visualize. Java offers many techniques that can be used to serialize java objects to XML or vice-versa. One of them is XMLEncoder and XMLDecoder which are classes in the java-beans package [21]. Another way is JAXB (Java Architecture for XML Binding) which provides a mechanism to marshal (write) java objects into XML and unmarshal (read) XML into an object [22].

3. Methodology

3.1. Overview

In this research, we introduce a new CEP query language CEP-ML and a language parser which comes with a web-based try-out tool. This chapter describes the methodology employed in developing CEP-ML which acts as a general CEP query language and the methodology of developing the language parser which converts from existing CEP languages to CEP-ML and vice versa. Finally, the try-out tool web UI which provides a simple user interface to construct CEP-ML queries and convert a given CEP-ML query to a preferred language and vice versa.

3.2. CEP-ML – Meta-language for CEP query languages

CEP-ML is a meta-language which acts as a general query language when interoperating between heterogeneous CEP engines. It can be used as an intermediate language to communicate with different CEP engines by converting from one language(a) to meta-language (CEP-ML) and then to another language(b).

CEP-ML is an XML based language with a defined schema. The language supports main CEP query operators which are frequently and commonly used in the industry. Refer Appendix A for a full explanation of the language syntax, schema and semantics.

3.2.1. Structure of the language

CEP ML possess an XML structure with xml tag names equal to operator keywords. Its XML structure provides several advantages. Since this language will be used as an intermediate language XML format is easier to convert to java object model and backwards in parser data flow. Also, if users want to generate CEP-ML queries they can use the web try out tool or implement their own programs easily using existing standard XML generation methods.

CEP-ML language has a tree structure inherited by XML characteristics. Element keywords are a combination of CEP ML specific set of keywords and common other language keywords. Every query starts with root tag <query>. Projections are declared within <select>, where you have to define what attributes, or attributes with aggregate functions applied need to be selected from the stream. Then need to define the streams that the query is running on within <from> tag. Here any number of streams can be given as conjunction. Each <stream> data can be filtered with <filter> and also by defining window function within <window>. Data filtering conditions are defined using <where>. Other grouping on condition is defined with <group-by> and <having> elements. Finally, the optional <insertInto> defines the stream that resulting events need to be inserted. Following figure 3.1 shows a sample query which describes the main structure of language.

```
<query>
<select>
  <attributes>
    <attribute as="a1">att1</attribute>
  </attributes>
  <functions>
    <function func="function" as="funcAs">functionApplyParam</function>
    ...
  </functions>
</select>
<from>
  <streams>
    <stream as="as">
      <name>stream name</name>
      <window func="function_name">
        <parameters>
          <parameter>window parameter</parameter>
          ...
        </parameters>
      </window>
      <filter>stream_filter_condition</filter>
    </stream>
    ...
  </streams>
</from>
<where>..condition..</where>
<group-by>grp_by_param</group-by>
<having>grp_condition</having>
<insertInto>..insert-stream..</insertInto>
</query>
```

Figure 3.1 Structure of CEP-ML

Keywords such as query, stream. are specific to CEP ML. Others such as select, from, group-by, having, where ...etc are common keywords used in most of the existing languages and exist in CEP ML also with same keywords. Window function is represented in different ways in different languages and in CEP ML it is represented with ‘window’ keywords as in SiddiQL and EPL. Filter and InsertInto functions are present in SiddhiQL and CEP ML also support these functions under the same keywords.

3.2.2. CEP-ML operators

Currently, CEP-ML language supports query functions such as projections with SELECT and aggregate functions, define data analysing streams with FROM and joined streams with JOIN, data filtering of an individual stream with WINDOW and FILTER and joined streams with WHERE, grouping results on conditions with GROUP-BY and HAVING, inserting results into another stream with INSERT-INTO. Following samples shows how main operators are presented in CEP-ML.

Projection of attributes with condition - <select> with <filter>

Projection is for extracting only part of the information contained in the event. With <filter> we can filter out the number of selections based on a condition given. Filter conditions can be defined for each stream. This filter function acts similar to SiddhiQL filters. All the filtering conditions for that stream need to be defined within single <filter> element.

Following query in figure 3.2 describes how to select rooms and their location where room number is greater than 100 from the given stream ‘tempetarureStream’.

```
<query>
  <from>
    <streams>
      <stream>
        <name>tempetarureStream</name>
        <filter>roomNo>100</filter>
      </stream>
    </streams>
  </from>
  <select>
    <attributes>
      <attribute as="rm">roomNo</attribute>
      <attribute>location</attribute>
    </attributes>
  </select>
</query>
```

Figure 3.2 Projection with conditions

Filter events within a Windows - <window>

Window allows capturing a subset of events based on criteria from input event stream for calculation. Many languages have inbuilt window functions. For an example, SiddhiQL and EPL have time, length, cron, sort, frequent, unique etc and CQL has range and now window functions. In CEP-ML window is represented as <window> tag, in-built functions are can be declared as attribute 'func'. Other parameters are defined under <parameter> tag. Following query in figure 3.3 describes how to select all the events in 'temperatureStream' within last 5 min.

```
<query>
<select all="true"/>
<from>
  <streams>
    <stream>
      <name>tempetarureStream</name>
      <window func="time">
        <parameters>
          <parameter>5 min</parameter>
        </parameters>
      </window>
    </stream>
  </streams>
</from>
</query>
```

Figure 3.3 Window Filters

Group-By - <group-by> and Having - <having>

Group by allows us to group the aggregation based on the given group by attributes. Having allows us to filter events after aggregation and after processing at the selector. In CEP ML these are represented by <group-by> and <having> tags. If there are more than one group by variables those needs to be defined comma separated.

Following query in figure 3.4 describes how to find the max temperature per room where the temperature is more than 40 degrees.

```
<query>

  <select>
    <attributes>
      <attribute>roomNo</attribute>
    </attributes>
    <functions>
      <function as="maxTemp" operator="max">temperature</function>
    </functions>
  </select>

  <from>
    <streams>
      <stream>
        <name>tempetarureStream</name>
      </stream>
    </streams>
  </from>

  <group-by>roomNo</group-by>
  <having>temperature > 40</having>

</query>
```

Figure 3.4 Grouping with conditions

Joining more streams with Conjunction - <from>

Conjunction allows merging two event streams based on a condition.

Following query in figure 3.5 describes how to find location and temperature from joined streams traffic_stream and temperature_stream with filtering records of each

stream separately under <filter> and considering combined conditions also under <where>.

```
<query>

  <select>
    <attributes>
      <attribute as="loc">location</attribute>
      <attribute as="temp">temperature</attribute>
    </attributes>
  </select>

  <from>

    <streams>
      <stream as="trs">
        <name>traffic_stream</name>
        <window func="time">
          <parameters>
            <parameter>5 min</parameter>
          </parameters>
        </window>
        <filter>ts.level > 5</filter>
      </stream>

      <stream as="tps">
        <name>temperature_stream</name>
        <window func="time">
          <parameters>
            <parameter>10 min</parameter>
          </parameters>
        </window>
        <filter>tps.temp > 30</filter>
      </stream>
    </streams>

  </from>

  <where>trs.eventId = tps.eventId</where>
</query>
```

Figure 3.5 Conjunction

Aggregate Functions on projection attributes- <functions>

Aggregate functions can be used within projections. Different CEP languages support different inbuilt aggregate functions such as, sum, average, max, min, range...etc. In CEP-ML these functions can be declared as 'operator' attribute.

Following query describes how to select the maximum temperature per room and machineID for the last 5 min.

```
<query>
<from>
  <streams>
    <stream>
      <name>tempetarureStream</name>
      <window func="time">
        <parameters>
          <parameter>5 min</parameter>
        </parameters>
      </window>
    </stream>
  </streams>
</from>
<select>
  <attributes>
    <attribute>roomNo</attribute>
  </attributes>
  <functions>
    <function as="maxTemp" operator="max">temperature</function>
  </functions>
</select>
<group-by>roomNo, machineID</group-by>
</query>
```

Figure 3.6 Aggregation functions

3.2.3. CEP-ML syntax compared with other languages syntax

CEP-ML is designed considering one of the main factors which is making it more readable, simple and familiar to the existing CEP query language users by keeping language syntax keywords, ordering, grouping similar to existing languages. XML tag names of CEP-ML are mostly similar to the query keywords of other languages. Even though the ordering of the main elements under root <query> element is not affected and forced while parsing CEP-ML to other languages, it is recommended to maintain the natural order similar to other languages to increase the readability.

Table 3.1 shows a comparison of how a sample CEP-ML query is represented in EPL, SiddhiQL and CQL languages.

Table 3.1 Comparison between language query syntax

Language	Syntax

CEP-ML	`<query>`
	`  <select>`
	`  <attributes>`
	`  <attribute as="attr1as">attribute1</attribute>`
	`  </attributes>`
	`  <functions>`
	`  <function func="funnc" as="attr2as">attribute2</function>`
	`  </functions>`
	`  </select>`
	`  <from>`
	`  <streams>`
	`  <stream as="st1">`
	`  <name>stream1</name>`
	`  <window func="windowFunc">`
	`  <parameters>`
	`  <parameter>parameter1</parameter>`
	`  </parameters>`
	`  </window>`
	`  </stream>`
	`  <stream as="st2">`
	`  <name>stream2</name>`
	`  </stream>`
	`  </streams>`
	`  </from>`
	`  <where>where_condition</where>`
	`  <group-by>group_by_variables</group-by>`
	`  <having>having_condition</having>`
	`</query>`

EPL	**select** attribute1 **as** attr1as, **func**(attribute2) **as** attr2as **from** stream1.**win**:**windowFunc**(parameter1) **as** st1, stream2 **as** st2 **where** where_condition **group by** group_by_variables **having** having_condition
SiddhiQL	**from** stream1**#window.windowFunc**(parameter1) **as** st1 **join** stream2 **as** st2 **on** where_condition **select** attribute1 **as** attr1as, **func**(attribute2) **as** attr2as **group by** group_by_variables **having** having_condition
CQL	**select** attribute1 **as** attr1as, **func**(attribute2) **as** attr2as **from** stream1[**windowFunc** parameter1] **as** st1 , stream2 **as** st2 **where** where_condition **group by** group_by_variables **having** having_condition

As shown in Table 3.1 and Table 2.4, Some query functions and their syntax are exactly similar in all 3 languages and CEP ML also has maintained a similar syntax in xml tags. Those functions are projection with query words 'select' and 'as', aggregate functions with query words 'avg, max, min.. etc' , grouping on a given condition with query words 'group by' and 'having' etc.

There are set of functions that are supported in all 3 languages but with different syntax and different depths. Conjunction is represented in EPL and CQL with a comma where in SiddhiQL it is defined with the keyword 'join'. CEP ML does not use any special keyword for joining streams and any number of streams need to be joined can be defined under <streams> tag. Even though EPL language supports advanced joins such as LEFT and RIGHT joins, those are not supported in current CEP ML implementation. Filtering with window is supported in all 3 languages but EPL and SiddhiQL have similar syntax and window functions where CQL differs in syntax and supports only limited and different window functions. CEP ML use <window> tag and has the ability to define any window function belongs to any of the languages.

Since EPL and SiddhiQL are DS type query languages, those support inserting results into another stream with 'insert into'. CEP ML also have introduced this with tag <insert into>. Filtering from individual streams is also only supported in EPL and SiddhiQL by using '[ ]' after stream definition and in CEP ML this feature is supported within <filter> tag within each stream.

3.2.4. CEP-ML as a mediator language

CEP ML is a meta-language that can be used as an intermediate language when converting between different languages. Rules that need to be shared across heterogeneous CEP can be written in any existing language, and CEP ML language parser can be used to convert that rule to CEP ML and then to other CEP engine rule language type so that user does not have to know the target language syntax.

CEP ML also can be used as a mediator language while migrating over different CEP engines. Currently, not having a language parser makes it difficult to migrate to new CEP engines from existing. Manual query conversion is a hectic process and need extra resources and time. CEP ML language parser makes this process easier since users can directly convert between languages. In this occasion CEP ML act as an intermediate language format.

3.3. Language parser

This section illustrates the methodology employed in developing the CEP language parser. Its main responsibility is to translate existing CEP languages to CEP XML language and vice versa. Currently, it supports three existing languages EPL, SiddhiQL and CQL. Parser is packaged as a java API which can be used in CEP engines integration systems implemented in java. It also exposes its API functionalities as a REST API with the try-out tool comes with it which will be discussed in coming sections.

In this research project scope, the languages that parser supports belongs to Data stream query (DS) and Composition-operator-based (CO) language types as explained in chapter 2. Siddhi and EPL belong to DS language type have SQL type query language and support common features including time window functions. CQL of Stream is classified as both DS and CO and also have a query structure similar to SQL.

3.3.1. Parse languages using CEP-ML language parser

Query parser can mainly intake CEP-ML queries and other language queries. Depending on the query type, users can select the converter. Each converter provides utilities to convert to CEP-ML or to another query type. Then comes the model creator to create the intermediate query model by parsing the input. Here CEP-XMLs are converted using standard JAXB and other queries are converted using query parser which will be discussed later. Once the model is created query printer is used

to construct queries and JAXB to generate CEP-ML. Figure 3.7 describes the outline of the parser implementation.

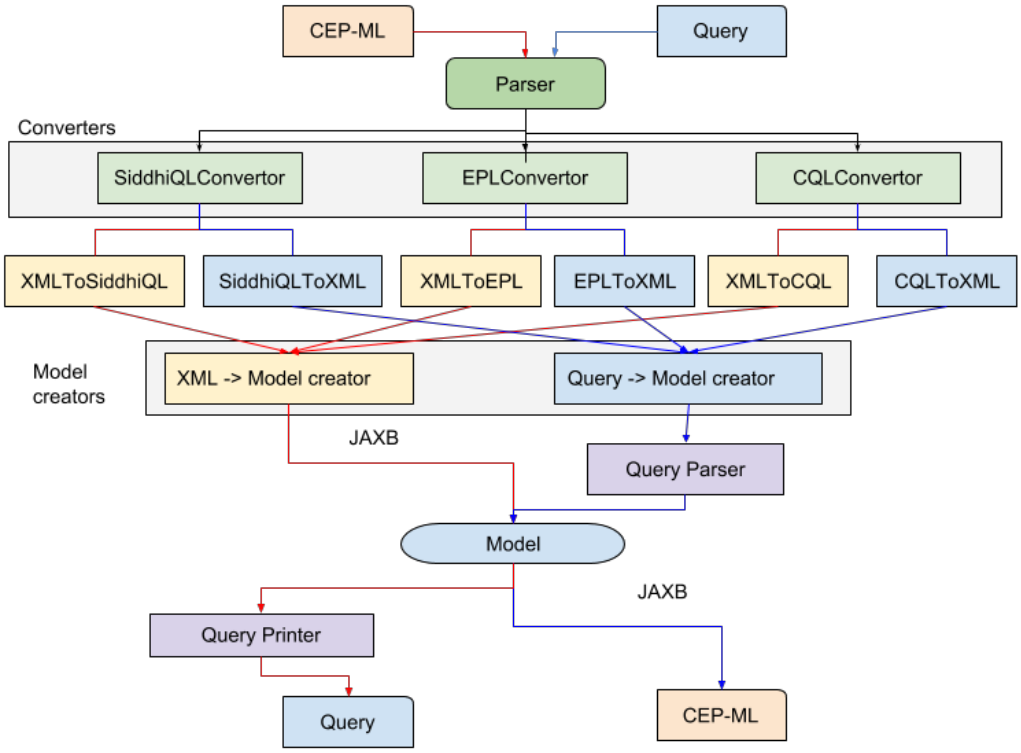


Figure 3.7 Outline of language parser

3.3.2. Query Parser and Query Printer

Query parser is responsible for parsing the query into the intermediate model object. Since query syntax and keywords will differ based on the type of the query (SiddhiQL, EPL or CQL) query string will be analysed and parsed to extract information. The following Algorithm 1 explain how a given query is converted to the intermediate query model. Here, a given query string is split by keywords and values are extracted and construct the model object.

```
Function Query<A>ToModel (q)

Input: Query string q of query type A
Output: Query Model o

Create Empty Model o

// setting 'from-stream'
s ← keyword for from-stream of query A
valueBlock ← get string block between given s and next keyword
from q
 // next keyword is one of pre-defined keywords set belongs to
that query type A
Value ← extract actual value from valueBlock
obj.fromStream ← Value

//setting 'select'
s ← keyword for select of query type A
valueBlock ← get string block between given S and next keyword
 // next keyword is one of keywords belongs to that query type A
attribute _list ← extract attributes from valueBlock
Value ← extract actual value from valueBlock
Create new Select object s
s.attributes ← attribute_list
obj.select ← s

//set window
….

Return obj
```

Algorithm 1 - Given Query to intermediate model object conversion algorithm

Query printer is responsible of converting a query model object to a requesting query string of type SiddhiQL, EPL or CQL. Following Algorithm 2 in pseudo code explain the algorithm used in query printer. Here based on the type of the query language need to be generated the syntax printing order is defined. Then query for the required language is constructed by using data in query model object constructed while parsing CEP ML.

```
Function Print<A>Query(obj)

Input: Query object obj
Output: Query q of type A

Create empty query string q
//generate 'from-stream' part
q ← append from-stream keyword of query type A
v ← get from-string value from obj
q ← append v to q

// read values of sub objects and append to query
// create 'select' values part
q ← append select keyword of query type A
atts ← obj.getSelectObject.getAttributes
q ← append attributes

// create window part
...
```

Algorithm 2 - Convert query model to existing query

3.3.3. Language parser implementation

Language parser is designed following model-driven architecture. It uses an intermediate data model to carry data while converting from CEP ML to other languages and vice versa. Figure 3.8 describes the main architecture of language parser with its components and data flows in compile time and run time with regards to translating XML to textual notation and vice versa.

Language parser is implemented in java language and JAXB is used for marshalling and unmarshalling Java objects to XML and vice versa which is the standard solution in java. Complete java doc for parser API is included in Appendix C.

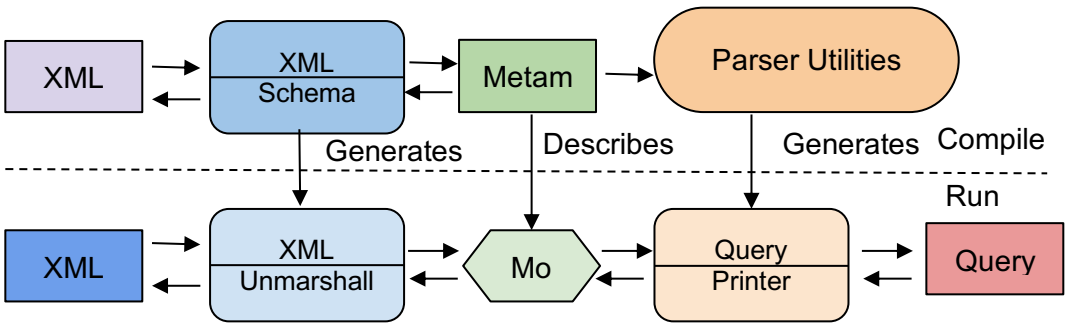


Figure 3.8 Model Driven architecture of language parser

3.3.4. Query Model

Query Model act as an intermediate data structures while converting XML to queries and vice versa. These are java objects annotated with JAXB annotations which maps xml structure to java objects. The root model is 'Query' shown in Figure 3.9 below. Find the full model class diagram in Appendix C.

```
@XmlRootElement(name = "query")
@XmlAccessorType(XmlAccessType.FIELD)
public class Query {
  @XmlElement(name = "select")
  private Select select;
  @XmlElement(name = "insertInto")
  private String insertInto;
  @XmlElement(name = "group-by")
  private String groupBy;
  @XmlElement(name = "having", type = String.class)
  private String having;
  @XmlElement(name = "from")
  private From from;
  @XmlElement(name = "where")
  private String where;
…
}
```

Figure 3.9 JAXB Query model

3.4. WebUI

CEP-ML is presented along with a web based try out tool. The main purpose of this tool is to try out the language parser API methods with a visual interface. This is capable of constructing CEP-ML queries and converting them to other preferred languages and vice versa.

This tool is a simple platform independent web application packaged as a WAR which can be deployed in any web application server. Front end user interface is implemented using ExtJS6 and backend rest API is implemented using SpringBoot and JAVA.

It also exposes a REST API which can be invoked from remote platforms as well. This enables users to use the parser as a web-service instead of java library. Following figure 3.10 is a sample view of the UI. More sample use cases are illustrated in Appendix E.

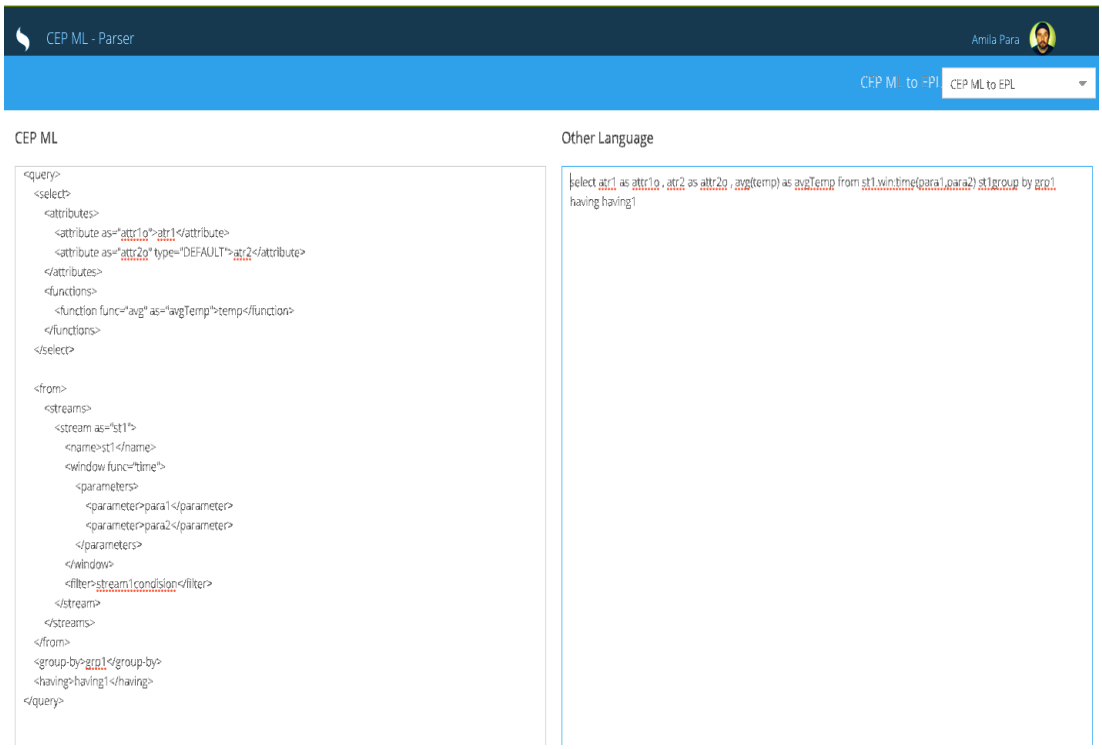


Figure 3.10 CEP ML Try-out web UI

This tool implementation follows MVC architecture. View component is implemented using Ext-JS where controller and service layers are implemented using spring-boot-web. View component allows users to insert any query and select the preferred language type they need to convert that query to. These conversion requests are sent to the controller in JSON format as REST requests. Controller converts these requests to java objects and pass to the service layer to generate the converted query and pass the response back to the view component. Service layer does the language conversion by calling the parser java library.

Discussion

We were able to develop a new CEP query language, CEP-ML to support interoperability of CEP engines. This language supports common query operators and it is readable and extendable. Also, to convert other existing languages queries to CEP-ML and vice versa we have come up with a language parser which is presented as a JAVA API. It also comes with a web based try out tool with a user interface where users can easily try constructing CEP-ML queries from other language queries and vice versa. Try-out tool also exposes a REST API for language conversion functions. Figure 3.11 shows the component diagram of the complete system. Complete project source code is available in repository [22].

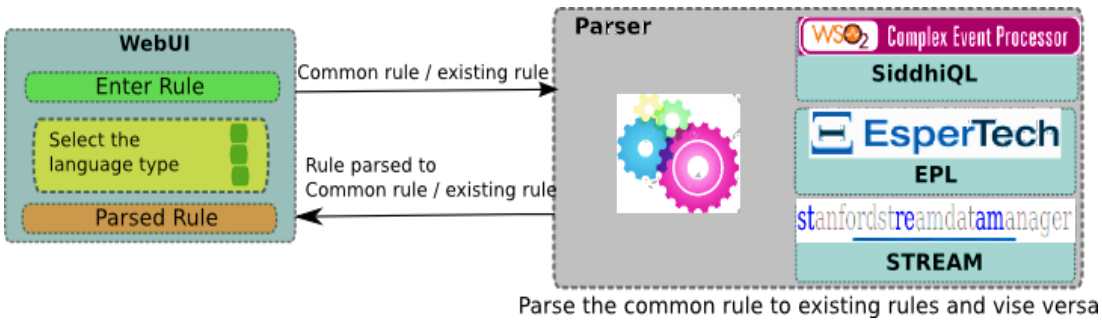


Figure 3.11 CEP-ML complete system component Diagram

4. EVALUATION

4.1. Overview

In this study, we perform the analysis of CEP-ML language and language parser's characteristics and expressibility. The system is designed and implemented focusing on readability, understandability, extensibility and platform independability. In this section, we discuss how these characteristics are achieved in each of the modules of the system.

4.2. Evaluating characteristics of CEP-ML system

4.2.1. Readability and understandability

High readability and understandability are essential characteristics of a meta-language. That will impel the more users to use this the language. Even though language parser can handle the intermediate language structure while converting between query languages, having a clear, readable language increase the visibility and helps to rectify issues that might face during the integration of these systems. Following are the characteristics of CEP-ML that satisfy above.

1. XML based language

   XML has a well-known structure that is easily readable by human as well as other programs. There are many other tools or other programs already available to parse and view XML. Also, XML can define a schema which can be used as language definition where users have the full visibility of all the features and correct syntax of the language.

2. Having common keywords similar to existing language keywords.

   By maintaining language keywords similar to other language keywords, users can easily predict the meaning of the language syntax.

4.2.2. Extensibility

Language extensibility or the ability to add or modify the content models with a minimal impact is important for it to evolve fast. Within the scope of this research, we have only covered the main functionalities of CEP queries and there are more functions to be added. In future as more CEP engines come, more language types with different functionalities will come. Since CEP-ML is a general meta-language, it should have the capability to adapt for these changes easily. CEP-ML has a XML based structure which is easily modifiable by adding, deleting or changing the order of XML elements tree branches. Hence it has a good extensibility.

CEP-ML language parser is implemented following a model based architecture which makes it easy to add, remove or modify existing model variables and change relationships between sub-models as the XML structure is changed.

4.2.3. Platform independent language parser and tools

CEP-ML language parser is packaged as a java library which is platform independent. It only requires a JVM to be installed to run and can be imported as a jar dependency to any java based program.

Try out tool is a spring based web application where users can deploy in any popular application containers. It also gives the ability for users to use language parser's API methods via REST interface. This also ensures that the system is platform independent.

4.2.4. Expressibility of CEP-ML

Following Table 4.1 shows a comparison between the functions supported by CEP ML over RuleML and other 3 languages we consider in this research SiddhiQL, EPL and CQL (Refer table 1 for symbolic meanings).

Table 4.1 Expressibility of CEP - ML language

	EPL	SiddhiQL	CQL	RuleML	CEP-ML
Projection	+	+	+	-	+
Filter by windows	+	+	⊖	-	+
Filtering (filter/where)	+	+	+	+	+
Aggregation by group/ group by	+	+	+	-	+
Counting	+	+	+	-	+
Aggregation functions e.g.: min, max	+	+	+	-	+
Event instance consumption/ insert into	+	+	+	-	+
Conjunction, Join	+	+	+	⊕	⊕
Sequences	+	⊕	⊖	-	-
Disjunction	+	+	+	⊕	-
Using external data sources	+	⊕	⊖	-	-
Rule Parts (if, else, then, do)	-	-	-	+	-

As presented in Table 4.1 above, CEP ML supports most of the basic and common functions such as projections, filtering, windows, aggregation by group, aggregate functions, counts and inserting to other streams. It partially supports stream joins as it does not support for left/right joins. Currently, it doesn't support operators such as sequences, disjunctions, patterns, subqueries and using external data sources.

When comparing with Rule ML, CEP ML also follows a similar structure to represent the language which is XML. CEP ML xml tags are more aligned with keywords of existing languages because it is facile for the users to define new queries in their applications to be converted to other languages later with the use of CEP ML language parser.

Discussion

CEP ML is a readable and understandable CEP meta-language because of the characteristics inherited from XML and since it is composed of keywords which are similar to other existing languages. CEP ML language and its language parser are easily extendable due to the XML and model-based architecture used in the implementation. Language parser is easy to integrate as it is platform independent and it also provides a rest full API along with a try-out tool. CEP ML supports most common CEP query operators and there are more functions to be added in the future.

5 CONCLUSION AND FUTURE WORK

With the increasing size of data, CEP systems need to be distributed. In distributed complex event processing systems heterogeneity of the CEP engines that use their own query languages has become a major problem to the interoperability between them. To overcome this problem, a common meta-language and a method to parse those queries to the existing languages is an utter requirement. There are existing solutions that have tried to access this problem, but they are not up to the level that can be used in the industrial level.

In this research, the existing languages were analysed and categorised to find out the expressibility of different query types by each language. Based on that a new XML like meta-language is defined which supports the main query functionalities of CEP languages. Also, a language parser for three selected languages SiddhiQL, EPL and STREAM to convert in between this meta-language and selected languages is presented which can be extended in future to support other languages as well. The web based try out tool we present is helpful for users to construct new queries and try out the converting between CEP-ML to other languages and vice versa.

As future work, this language needs to be improved to make it more sophisticated in following areas.

- Support more CEP query functions such as Patterns, disjunctions etc. Even though some functions are specific to some query languages and not very common in all languages, as a meta-language, CEP ML language should be extendible to support these functions as well.
- Improve the parser to support more CEP languages.

  Currently, it supports only three popular CEP languages EPL, SiddhiQL and CQL and needs to improve to support more languages.

- The parser is presented as a java API library. Hence, can be integrated only with platforms which supports java. In future, this need to improve to support other language platforms as well such as C/C++
- Improve the language parsing logic in the parser to improve the performance

REFERENCES

[1] Björn Schilling, Boris Koldehofe, Udo Pletat and Kurt Rothermel, “Distributed Heterogeneous Event Processing Enhancing Scalability and Interoperability of CEP in an Industrial Context.” Proceedings of the 4th ACM International Conference on Distributed Event-Based Systems (DEBS). Cambridge, United Kingdom,2010

[2] M. Eckert and F. Bry, "Complex Event Processing (CEP)," in Institut f¨ur Informatik, Ludwig-Maximilians-Universit¨at M¨unchen. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.656.2988&rep=rep1&type=pdf. Accessed: Mar. 26, 2016.

[3] Schultz Møller, Nicholus Poul, Matteo Migliavacca & Pietzuch Peter. (2009). Distributed complex event processing with query rewriting. Proceedings of the Third ACM International Conference on Distributed Event-Based Systems DEBS 09, 1. http://doi.org/10.1145/1619258.1619264

[4] H. Boley, A. Paschke, and O. Shafiq, "RuleML 1.0," The Overarching Specification of Web Rules.

[5] Poul Schultz-Møller, Nicholas & Migliavacca, Matteo & R. Pietzuch, Peter. (2009). Distributed Complex Event Processing with Query Rewriting. Available : https://lsds.doc.ic.ac.uk/sites/default/files/debs09-next_ced_0.pdf. Accessed: Jan. 16, 2019.

[6] A. Demers, J. Gehrke, M. Hong, B. Panda, et al. Towards Expressive Publish/Subscribe Systems. In EDBT, 2006.

[7] A. Demers, J. Gehrke, M. Hong, B. Panda, et al. Cayuga: A Genaral Purpose Event Monitoring System. In CIDR, pages 412–422, 2007.

[8] P. R. Pietzuch, B. Shand, and J. Bacon. A Framework for Event Composition in Distributed Systems. InMiddleware, Rio de Janeiro, Brazil, jun 2003.

[9] S. Suhothayan, K.Gajasinghe, I.L. Narangoda, S. Chaturanga, S. Perera, V. Nanayakkara (2011). "Siddhi: A second look at complex event processing

architectures." Proceedings of the 2011 ACM workshop on Gateway computing environments.ACM, 2011. [Online]. Available: https://dl.acm.org/citation.cfm?id=2110493

[10] Schilling, B, Pletat, U & Rothermel, K. (2009). Event Correlation in Heterogeneous Environments Ereigniskorrelation in heterogenen Umgebungen. It - Information Technology, 51(5), 270–275. http://doi.org/10.1524/itit.2009.0551

[11] Hai-Lam. Bui, "Survey and Comparison of Event Query Languages Using Practical Examples," Ludwig Maximilian University of Munich (March 2009) [Online]. Available: http://www.en.pms.ifi.lmu.de/publications/diplomarbeiten/Hai-Lam.Bui/DA_Hai-Lam.Bui.pdf. Accessed: Mar. 26, 2016.

[12] EsperTech, “Products - Esper.” [Online]. Available: http://www.espertech.com/products/esper.php. [Accessed: 24-Jan-2016].

[13] "ESPER," in EPL. [Online]. Available: http://www.espertech.com/esper/release-5.2.0/esper-reference/html/epl_clauses.html. Accessed: Dec. 10, 2018.

[14] "WSO2 - Complex Event Processor," in SiddhiQL Guide 3.0, 2015. [Online]. Available: https://docs.wso2.com/display/CEP400/SiddhiQL+Guide+3.0#SiddhiQLGuide3.0-IntroductiontoSiddhiQueryLanguage. Accessed: Jan. 30, 2016.

[15] R. Motwani et al., "Query Processing, Resource Management, and Approximation in a Data Stream Management System," Stanford InfoLab Publication Server. [Online]. Available: http://ilpubs.stanford.edu:8090/549/1/2002-41.pdf. Accessed: Feb. 12, 2016.

[16] Paschke, Adrian. (2014). Reaction RuleML 1.0 for Rules, Events and Actions in Semantic Complex Event Processing. [Online]. Available: https://www.researchgate.net/publication/263125416_Reaction_RuleML_10_for_Rules_Events_and_Actions_in_Semantic_Complex_Event_Processing. Accessed: Jan. 02, 2019.

[17] A. Arasu, S. Babu, and J. Widom, The CQL Con tinuous Query Language: Semantic Foundations and Query Execution, Stanford University. [Online]. Available: http://ilpubs.stanford.edu:8090/758/1/2003-67.pdf. Stanford InfoLab Publication Server. Accessed: Mar. 21, 2016.

[18] R. Kajic, "Evaluation of the Stream Query Language CQL," UPPSALA University. [Online]. Available: http://www.it.uu.se/research/group/udbl/Theses/RobertKajicBSc.pdf. Accessed: Mar. 26, 2016.

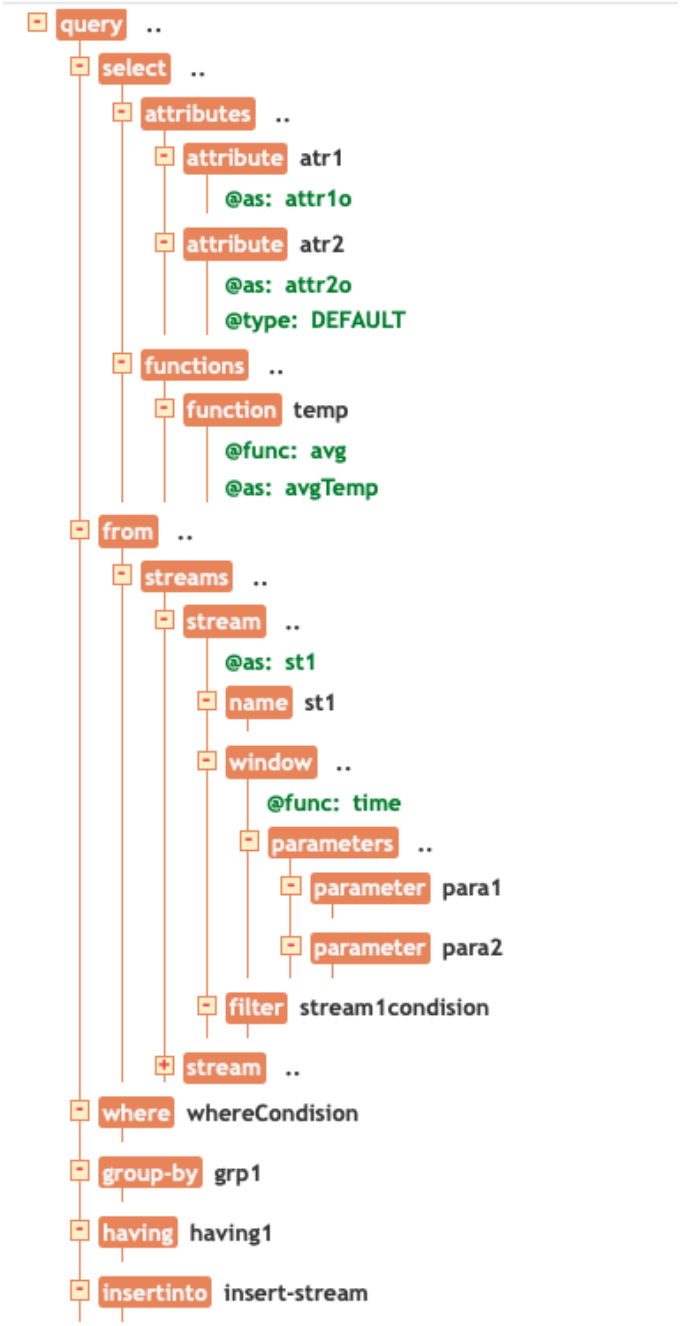
[19] Novak, Marek. (2010). Easy Implementation of Domain Specific Language using XML, Dept. of Computers and Informatics, FEI TU of Košice, Slovak Republic. [Online]. Available: https://www.researchgate.net/publication/228458637_Easy_Implementation_of_Domain_Specific_Language_using_XML

[20] Using XMLEncoder. [Online]. Available: https://www.oracle.com/technetwork/java/persistence4-140124.html . Accessed: Nov. 30, 2018.

[21] Java Architecture for XML Binding (JAXB). [Online]. Available: https://www.oracle.com/technetwork/articles/javase/index-140168.html . Accessed: Nov. 30, 2018.

[22] CEP ML - https://github.com/amilaparanawithana/CEPRuleLanguage

Appendix A: Dom tree view of the CEP XML language



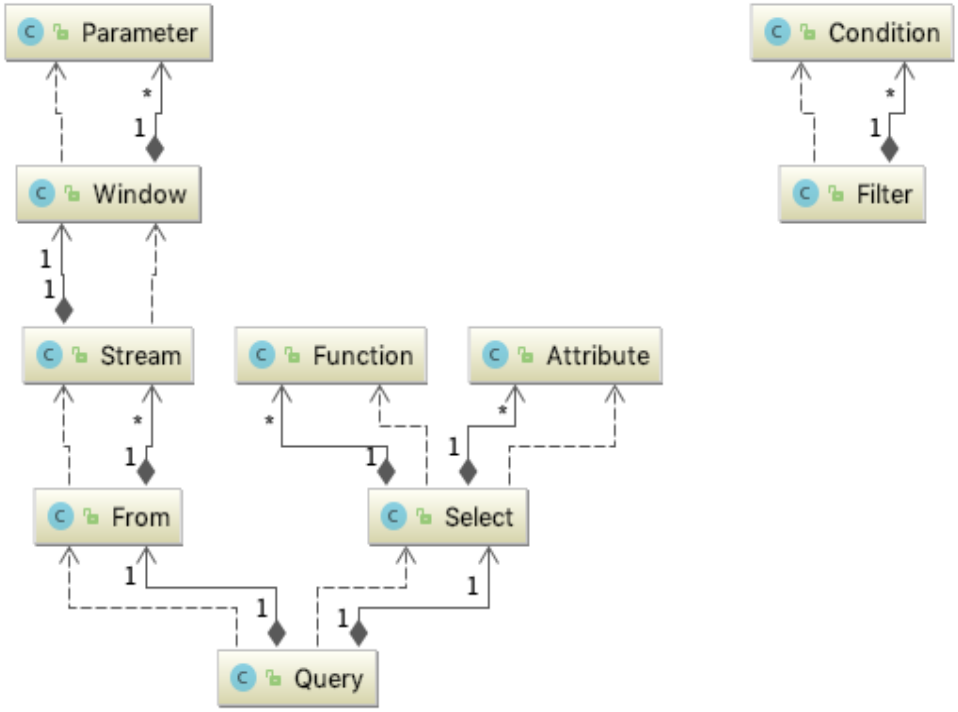
Appendix B: CEP XML language operations tags

SELECT	
syntax	<select> <attributes> <attribute as="attr1o">atr1</attribute> <attribute as="attr2o">atr2</attribute> </attributes> </select>
<select>	Define a select
<attributes>	Defines all the conditions
<attribute>	Define one condition Value - condition value
as	Result assign variable name
AGGREGATE FUNCTIONS	
syntax	<functions> <function func="function" as="funcAs">functionApplyParam </function> ... </functions>
<functions>	Define all selects applying functions
<function>	Define one function Value - function applying variable
func	Function type eg: max, avg
as	Result assign variable name
FILTER	
syntax	<filter>condition</filter>

<filter>	Define a filter Value - Filter condition
GROUP-BY	
syntax	<group-by>grp1</group-by>
<group-by>	Define a group by Value - Grouping columns comma seperated
HAVING	
syntax	<having>having1</having>
<having>	Define a having function Value - Having condition
WINDOW	
syntax	<window func="time"> <parameters> <parameter>para1</parameter> <parameter>para2</parameter> </parameters> </window>
<window>	Define a window function
func	Function type. Supported types are - Time - TimeBatch - Length - Cron
<parameters>	Window parameters
<parameter>	Define parameter Value - parameter value
INSERT INTO	
syntax	<insertInto>insert-stream</insertInto>

<insertInto>	Define result inserting stream Value - stream name
FROM and STREAMS	
syntax	<from> <streams> <stream as="as"> <name>stream name</name> </stream> </streams> </from>
<streams>	Defines all the input streams
<stream>	Define one stream. <window> <filter> tags also come under this.
<name>	Define stream name as value
as	Resulting stream assign variable

Appendix C: CEP ML implementation Models class diagram



Appendix D: Language Parser API methods

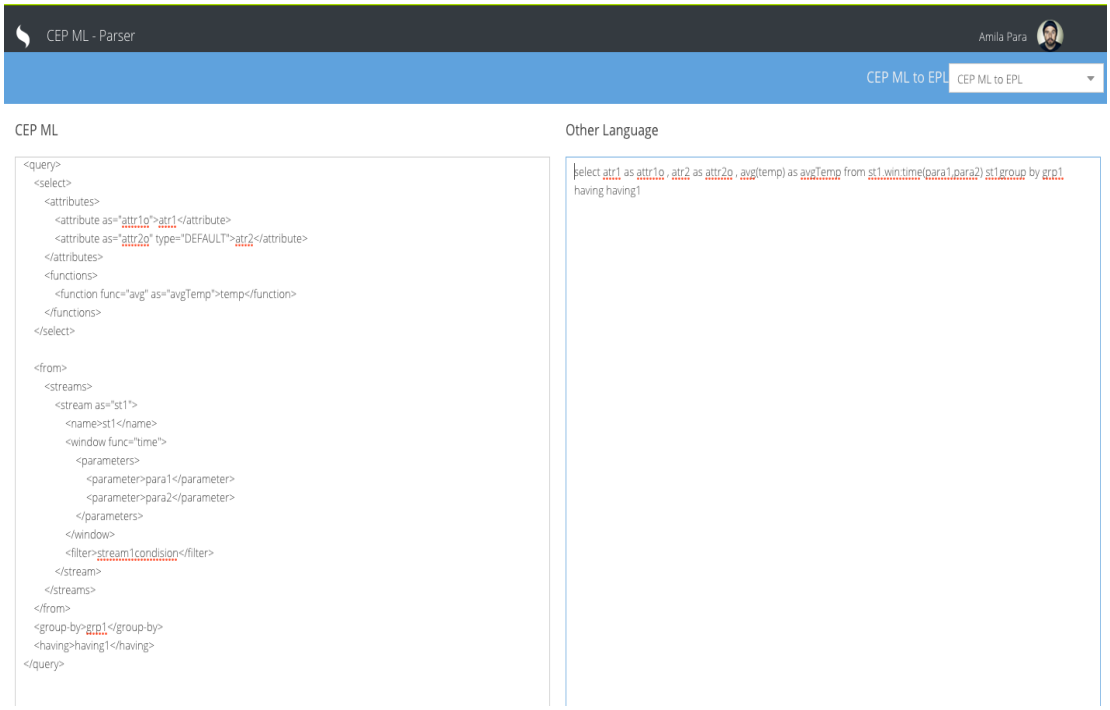
Method	Input	Output	Description
XMLToSiddhiQL	xml string	SiddhiQL query	Maps XML string to a Query object with jaxb and convert to a SiddhiQL
XMLToSiddhiQL	xml file	SiddhiQL query	Maps a XML file to a Query object with jaxb and convert to a SiddhiQL
SiddhiQLToXML	SiddhiQL query string	Siddhi query in CEP ML	Convert SiddhiQL query to CEP ML metalanguage
XMLToEPL	xml string	EPL query	Maps a XML string to a Query object with jaxb and convert to a EPL
XMLToEPL	xml file	EPL query	Maps a XML file to a Query object with jaxb and convert to a EPL
EPLToXML	EPL query string	Siddhi query in CEP ML	Convert EPL query to CEP ML metalanguage
XMLToCQL	xml string	Stream query	Maps a XML string to a Query object with jaxb and convert to a CQL
CQLToXML	CQL query string	CQL query in CEP ML	Convert CQL query to CEP ML metalanguage

Appendix E: CEP ML try-out tool

To convert CEP ML to other languages, put your CEP ML syntax in left side box and select CEP ML → <preferred language> from the drop down. Resulting query will be displayed in right side box.

Sample UI of converting CEP ML to EPL is shown below.

CEP ML to EPL in try-out tool



To convert from other languages to CEP ML, put your preferred language query syntax in right side box and select <preferred language> → CEP ML from the drop down. Resulting query will be displayed in left side box.

Sample UI of converting EPL to CEP ML is shown below.

EPL to CEP ML in try-out tool