Towards a REST-ful Visualization of Complex Event Streams and Patterns

Benjamin Kanagwa$^1$ and Nasser Kimbugwe$^2$
Makerere University, P.O.Box 7062, Kampala, Uganda
$^1$bkanagwa@cis.mak.ac.ug, $^2$nkimbugwe@cis.mak.ac.ug

Abstract—This paper presents a scalable architecture for visualization of complex events, event hierarchies and relations. The goal is to support flexible, multi-level, multidimensional visualization of primitive and composite events to aid faster decision making. Our approach is to apply the concept of Representational State Transfer (REST) that focuses on design of large scale distributed systems. Different visualizations are first and foremost mapped to state representations of Complex Event Processing (CEP) systems. We show how RESTful design can be applied to common CEP visualization requirements such as root-cause analysis. Finally, we present a prototype implementation of the API using a case study of event streams in public procurement.

Keywords: stream processing, complex event processing, visualization, Representational State Transfer (REST).

1. Introduction

Diversity and responsiveness desired by users of complex event processing systems to explore large datasets in respect to different aspects has created a need for a more generic and scalable visualization infrastructure. We propose application of Representational State Transfer (REST) [12] to visualization in Complex Event Processing [16], [22]. REST is an architecture style designed for large scale distributed systems and has been applied in design of web services [8], [9], and other applications including twitter$^1$ and facebook$^2$ among others. The goal is to design an architecture that brings REST’s advantages of loose coupling and good scalability to the discipline of CEP. The resulting uniformed interface provided by REST creates new opportunities including potential to integrated CEP systems across domains including mashups of CEP based systems. It has been noted by [21] that CEP is not yet fully exploited and is still establishing itself within the business world due to its lack of scalability and interoperability both within the event detection technologies as well as the visualization techniques. Current CEP visualizations do not provide the flexibility to alter, adopt and analyze the data from different perspectives.

CEP based systems continuously process events as they happen to infer complex relations based on context and temporal relationships. An event is any important occurrence that is worth noting and represents the instance of an activity [17]. Each organization has many events that happen at different times, some related and others not. Occurrence of events in time-space creates a continuous flow of events leading to the notion of event streams. An event stream has been defined as an infinite sequence of events [1]. Therefore, organisations have many event streams flowing into different information systems. CEP does not look at individual events but seeks to detect presence or absence of given event combinations (complex events) tied together by temporal relations. The event sources may be internal of external to the organization. External sources may include social media [3], [4] among others. The role of a CEP system is to continuously listen to incoming event streams and filter out desired combinations of events on which aggregation functions are applied to infer trends and instant insights for decision making.

Current users of CEP systems interact through dashboards with generic graphical representations for all users. However, users require personalized visualizations and appropriate access controls where a single user may be interested in different representations of the same event. It has been noted that visualization of these events in real-time is not yet scalable. Moreover current approaches make it impossible to integrate different CEP systems. This is largely due different event, pattern and visual representations. We note that provision of information is one aspect that must be complimented with appropriate representations that facilitate faster decision making. Personalized visual tools should allow browsing and exploration of events and related causes. We contend that each user at whatever level has some decision to make. Accurate, and current information is therefore required by users in varying level of detail and sophistication.

This problem is further compounded by large organizations and cloud environments, where thousands of users may be interested in observing different patterns. This requires the system to provide scalable mechanisms for serving such large number of patterns. Distributed scalable systems have been addressed by the Representational State Transfer (REST) design—a resource oriented architecture designed for scalable distributed systems. Through its uniform in-

---

1http://twitter.com
2http://facebook.com
terface, application of REST[12] to CEP brings several advantages. First, it enables scalable management of growing number of resources in a consistent way. Second, it provides a simple and flexible way to build CEP driven integrations and applications through a uniform programmable interface.

The focus of this paper is to provide a scalable architecture for visualization of data streams. In this paper we make the following contributions (i) we identify and categorize important resources for visualization in CEP (ii) define an API for scalable visualization of CEP patterns (iii) present a prototype implementation of the API. The rest of the paper is organized as follows -: in Section 2 we give related work followed by background in Section 3. Section 4 is a discussion of the design goals followed by the structure of the REST API in Section 5. We present a prototype implementation in Section 6 followed by a discussion and conclusion in Section 7.

2. Related Work

Many CEP tools and platforms have emerged in recent years. This is attributed to its wide applicability [19] including domains of finance, health care, fraud detection, and online business among others. Most notable platforms are IBM System S[7], Oracle[18] and Esper[7]. The Esper language and processing algorithm are integrated into the Java and .Net (NEsper) as libraries. These platforms differ at different levels including (i) internal representation of events - where a variety of alternatives exits including extensible Markup Language(XML), JavaScript Object Notation (JSON), internal object representations that are native to specific programming languages such as Java (ii) internal processing technologies - where two dominant options are Finite State Automata (FSA) based techniques and Event processing Networks ( EPN) (iii) the number of extent of visualization support provided.

Esper is one of the leading open source CEP engines where events are modelled as object instances that expose event properties through Java Bean-style getter methods. In Esper, event classes or interfaces do not have to be fully compliant to the Java Bean specification; however for the Esper engine to obtain event properties, the required Java Bean getter methods must be present. Event specification languages provide constructs to define relations between event streams. For instance, Esper has in built functions for pattern matching [11] ,which uses five operators that include the following:- every - Operators that control pattern finder creation and termination; Logical operators - or, and , not; Temporal operators - that operate on event order → (followed by): guards and where - conditions that filter out events and cause termination of the pattern finder. According to Esper [11] the Esper engine exceeds 500 000 event per second with engine latency below 3 milliseconds.

In [4], Samujjwal suggested an adaptive event stream processing (ESP) environment in which he explored the limitations of current ESP systems due to fixed pattern detection mechanism. However, his work was more on pattern detection than on the visualization of the events. Annett and Stroulia in [2] developed a REST application called Invenio that could geographically visualize aggregated music chart information. However, Invenio is not a CEP application.

3. Background

3.1 Representational State Transfer(REST)

Since the invention of the REST architecture by Roy[12], it has seen wide adoptability and practical implementations that include Twitter[6], Facebook[20], Google[13], [14] and Ruby on Rails [15] among others. The modern web is one instance of a RESTful-style architecture [12] although applications can include access to other styles of interactions. In most cases, the REST API is provided as means to allow developers who want to embed functionality into their applications. REST is an architectural style which is designed around the concept of a resource [12]. REST systems expose their data and functionality through resources identified by Uniform Resource Identifiers (URI). REST-style architectures consist of clients and servers. Clients make requests to servers and servers respond to their clients by acting upon each request and returning appropriate responses. Each request and response is considered as transfer of a representation of a resource - hence the name REpresentation State Transfer.

A resource is a conceptual mapping to a set of entities [12]. In practice, a resource is anything that is important enough to be referenced by itself. However, during interactions, resources are represented in an addressable format like HTML, XML, RSS, PDF and many others depending on the prevailing need. Interaction between clients and servers is constrained by uniform interfaces defined by a fixed set of verbs. For instance, Hyper Text Transfer Protocol (HTTP) based implementations use the standard HTTP methods: GET, HEAD, POST, PUT and DELETE. A uniform interface only offers a set of operations including ability to retrieve, change or create data among others. The simplicity of the uniform interface is important to REST, because it keeps the interaction between client and server as simple as possible.

Resources can be accessed via HTTP. For each resource, a unique identifier is provided and makes part of the Uniform Resource Locators (URL) for operations on the resource. With REST, developers are able to create mashup applications that aggregate numerous sources of information and promote rich user interaction [2]. As an example let us consider the Twitter API with the URL structure of http://twitter.com/statuses/user_timeline.extension.
3.2 Complex Event Processing

An event can be categorized as primitive or composite event [24]. A primitive event is directly observed in the system while a composite event is a concatenation of primitive events using event algebra. Primitive events also known as atomic occur instantaneously.

Fig. 1: Pattern matching(Adopted from [23])

CEP systems apply a query technique called pattern matching where a set of event streams is matched against a complex pattern that specifies constraints on extent, order, values, and quantification of matching events [5]. This is mostly done using a pattern query which addresses a sequence of events that occur in order and are correlated based on the values of their attributes [1]. Figure 1 shows streams InputStream1 and InputStream2 that combined to detect pattern Pattern1. Upon detection of Pattern1 an output Out is generated. Event stream queries provide the windows, aggregation, joining and analysis functions for use with streams of events[11]. As of today there is no standard pattern query language and each CEP engine uses either graphical interface, a scripting language or a combination of both. Pattern languages specify the constraints on the events to be detected and remain silent on how the resultant event instances are to be visualized.

4. Requirements and Design Goals

Our design is based on a set of design goals as listed below:-

R1 - Scalability: CEP systems may be deployed at a small entity or very large entity. Therefore the number of concurrent users may vary from tens to several thousands. Rendering of graphs is time and computational intensive. It is therefore required that a CEP visualization infrastructure be designed with scalability.

R2 - Abstraction: The ability to fold and unfold, zoom in and zoom out different pieces of data is very important. This entails showing finer details of a piece of data when requested. Also abstraction allows hierarchical representation of events for different management levels. Aggregation and causality are fundamental concepts in CEP. Aggregation defines which set of primitive events combine to form a given complex event. The complex event is seen here as an abstraction of the primitive events. Causality involves sets of events that happen because another set of events had to happen. Both relations can be ably explored using abstraction techniques.

R3 - Flexibility: The ability to easily change the representation of data to provide quicker interpretations. This requires ability to transform from one visualization to another. For instance a user may want to render pie-chart or tabular visualization of the same data.

R4 - Mashup: Provide a basis for creating applications that aggregate different primitive and complex event patterns from disparate stream processing platforms

5. The REST API structure

The key consideration is that the application for which the the API is being designed for exists independently of the API. We identify the states of the applications and operations that can be carried out.

5.1 Resources

As noted earlier, resources form the foundation of a RESTful design because interactions are based on transfer of resource representations. One approach used to identify states and resources is to look at resources as equivalent to entities in an Entity Relation (ER) diagram. Following this approach, we identify the resources indicated in figure 2. These base resources are then augmented with derived resources such as lists of events, lists of activities, causal sets (history) that are peculiar to CEP visualization.

Fig. 2: Relationship between Complex Event Processing Resources

- **Activity**: These are event templates that define the types of events expected in the CEP system. The activity provide the event type and specifies the attributes and features of the event.
- **Event**: These occurrences representing that a specific activity has happened
• **Pattern**: group of activities that are constrained to happen under certain conditions based on ordering, extent, values and quantification of matching set.

• **Causal Models**: The events that cause an event to occur are bound to it [24]. They provide the information about the event through the parameters of the participating events.

• **Abstraction Hierarchies**: These represent how activities aggregate for form high-level activities

• **History**: a partially ordered set of event instances that contributed to the occurrence of a given complex event.

5.2 URI

To illustrate the API, let $E_1, E_2$ be primitive event types. If events $E_1$ is required to happen before $E_2$ to match complex event $E_5$ then $E_1, E_2$ are components of complex event $E_5$.

5.2.1 Activity API

The general format for the activity API is 
http://{activities}/{id}/format={fmt} where fmt is one of bar, piechart, linegraph, tabular. This is further argumented with the HTTP methods as indicated in the table 1. The id represents the identity of the activity for which event instances belong.

<table>
<thead>
<tr>
<th>HTTP Method</th>
<th>URI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/activities/</td>
<td>Returns a list of activities in the system.</td>
</tr>
<tr>
<td>GET</td>
<td>/activities/id</td>
<td>Returns a specific activity with corresponding id.</td>
</tr>
<tr>
<td>POST</td>
<td>/activities/id</td>
<td>Posts a given activity</td>
</tr>
<tr>
<td>PUT</td>
<td>/activities/id</td>
<td>Updates the activity with the specified id</td>
</tr>
<tr>
<td>DELETE</td>
<td>/activities/id</td>
<td>Deletes the activity with the specified id</td>
</tr>
<tr>
<td>GET</td>
<td>/activities/id/</td>
<td>Returns a specific activity with corresponding id.</td>
</tr>
<tr>
<td></td>
<td>components</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: REST API for Event Resources. Retrieves both complex and primitive activities

5.2.2 Event API

The general format for the events API is
http://{events}/{id}/format={fmt}. This is further argumented with the HTTP methods as indicated in the table 2. The id represents the identity of the event.

Table 2 shows the events API. The last row is used to represent to retrieve the event history - the set of all events that led to its occurrence. Consider the events of type $E_1$ that trigger events of type $E_7$ and $E_2$ in turn triggers $E_9$, represented as $E_5 \rightarrow E_7 \rightarrow E_9$. When a match for $E_9$ is obtained then instances of $E_7$ and $E_5$ make up the history.

<table>
<thead>
<tr>
<th>HTTP Method</th>
<th>URI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/events/</td>
<td>Returns a list of events the system.</td>
</tr>
<tr>
<td>GET</td>
<td>/events/id</td>
<td>Returns a specific event with corresponding id.</td>
</tr>
<tr>
<td>POST</td>
<td>/events/id</td>
<td></td>
</tr>
<tr>
<td>PUT</td>
<td>/events/id</td>
<td></td>
</tr>
<tr>
<td>DELETE</td>
<td>/events/id</td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>events/id/</td>
<td>the results would be $e_1, e_2$</td>
</tr>
<tr>
<td></td>
<td>history</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: REST API for Event Resources. Retrieves both complex and primitive events

5.2.3 Zooming, Root cause analysis

The general format for the zooming API is 
http://(zoom)/{id}/format={fmt}. In this case the id represents either an activity or event. Complex events that result from pattern matching are also assigned a unique id and therefore can be accessed through the API. The zooming API is applicable to both an activity or complex event. Zooming into a complex activity retrieves details that related to the hierarchical structure of the activity.

6. Implementation

We implementad our own engine called KAMANDA that is based on Non-deterministic Finite Automaton (NFA)[10]. The KAMANDA interface represents patterns using a graphical interfaces as indicated in Figure 3(a). The event sources are captures by adapters that connect to databases or some instrumentation. The engine is implemented using Java where events are modeled as JSON objects. The visualization interface is web-based in which we implement the RESTFul API. Events are made to pass through a CEP engine which filters them depending on the conditions set. All events, either directly from the CEP engine or from Database, are visualized through a REST visualization API accessible through the web browser.

Unlike many stream processing engines that use a combination of scripting and graphic interfaces for pattern language, the KAMANDA platform uses only a graphical user interface. This choice is based on the need to place control of information in the hands of managers and executives who are normally not IT experts and therefore find scripting languages hard to deal with. The graphical user interface provides several defaults that can be adjusted to provide more expressive pattern specification options.

For illustration, we consider event streams $E_1$ and $E_2$ that represents publish and sale for an online auctioning system. The publish event represents an activity of a user publishing an item for sale, while the event sale represents an activity of sale of item to highest bidder. Figure 3 (a) shows how the pattern is represented. The conditions on
patterns are set through the expandable activity tabs. Figure 3 (b) shows the corresponding representations using a pie chart. In the same figure, a user can quickly choose between different representations. The segments in the pie chart is the total number of matching event patterns grouped every minute. So a number of 5 in the pie chart implies 5 pairs of \{publish, sale\} combinations.

We provide two levels of root cause analysis through zooming. The first zoom level explores the details of aggregated value by window. The second level takes the individual results to show the specific event instances that make the original aggregated value. From a graph segment say in piechart one can zoom to see further details which we show in Figure 4 (a). This is the first level of zooming. The second level of zooming will show the individual events as indicated in Figure 4 (b).

7. Conclusion

We have been able to identify different resources akin to complex event processing. The resources have been exposed through a RESTful API allowing visualization. Through the proposed RESTful API and implementation, we have been able to provide a consistent and flexible means of navigating event stream outputs. Third party developers can embed CEP outputs into their systems. Our next step is to increase the expressiveness of the graphical pattern language as well the number of visual representations.

References


