COGILITY

Overview of the Cogynt Event Stream Processing Platform (ESP) Analytic

A Powerful ESP Analytic Solution Built for Analysts

Introduction

Cogility has been developing analytic solutions for the past 10 years. A valuable lesson we've learned is that the best analysts don't usually have good programming skills, and good programmers don't usually have strong analyst backgrounds. Cogility's mission has been to close this gap by empowering analysts to more efficiently and effectively use their cognitive skills to probe data - without any coding dependencies. We believe that the Cogynt Event Stream Processing (ESP) Platform best serves this purpose.

The Cogynt ESP analytic platform utilizes a declarative[1], zero coding authoring capability that completely eliminates the need for programming. Cogynt ESP has application to all sectors: it effectively addresses a broad range of situation awareness and decision support use cases, employing Hierarchical Complex Event Processing (HCEP) as its analytic core. HCEP is a computational hierarchy that maintains the state of patterns for long-lived processes and is deterministic, i.e., allowing for human evaluation and assessment for all the granular events supporting system generated conclusions. Cogynt ESP allows analysts to hypothesize and define complex multi-layered behavioral patterns: this effectively reduces the proverbial haystack to a manageable size, as the data is continuously sifted to identify pattern components. In terms of human capital -specifically analysts- Cogynt ESP is a force multiplier that greatly improves analyst effectiveness and enables dramatic improvements in enterprise agility and proactive response to risks and capturing opportunities. Cogynt ESP thrives in time sensitive, complex environments.

As an IT platform, Cogynt ESP is a highly scalable and performant ESP solution. It leverages Apache Kafka as the messaging and persistence backbone, and Apache Flink as its stateful compute engine. Cogynt ESP effectively integrates these open source technologies, making them mostly transparent to the end user.

The primary purpose of this white paper is to describe the conceptual basis for Cogility's implementation of HCEP and its use of Event Pattern Constraint Language (EPCL), which provides the means for an analyst to define computationally complete hierarchical patterns without having to do any programming.

[1] https://en.wikipedia.org/wiki/Declarative_programming

Introduction to Complex Event Processing (CEP) – Identifying a Wedding
This section explains HCEP, which begins with a discussion of traditional CEP, and how it works. The wedding example, in Figure 1, is a good place to start. It is based on the example offered on the *Complex Event Processing wiki site*[2], which is easy to understand and visualize. In this example

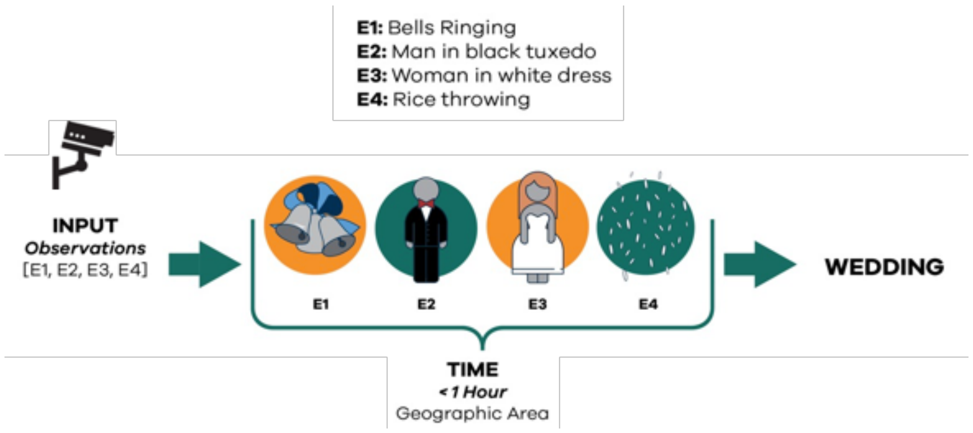


Figure 1 Basic CEP Example

we are interested in detecting a traditional western wedding. It is visually intuitive that this "event pattern" defines 4 specific events types (*church bells ringing*, *man in a tuxedo*, *woman in a white dress* and *rice in the air),* and 2 constraints (*time <1 hour* and specific *geographic area*), which in combination infer that a wedding has occurred. If a human being directly observed this pattern, the human could quickly "connect-the-dots" and determine that a wedding had occurred. For a computer-based system, this isn't so easy. First, we would have to assume that we had a smart analytic that could provide the appropriate object recognition; second, the computer would also have to correlate the observations to a set of boundary conditions - in this case, a time-window and precise geographic area, to conclude that a wedding has happened. Both the observations and the constraints must be valid to infer a wedding. This combined correlation process is the basis of complex event processing.

What if we want to connect these complex events to yet higher level patterns? Could we ask (hypothesize) more complex questions about the wedding, such as was whether the wedding reception was held at the same location as the wedding ceremony, or at a different location? What new patterns are needed to make this determination? This is where Hierarchical Complex Event Processing (HCEP) comes into play. It starts with a hypothesis, or a question to be answered and an understanding of how to detect the events needed to validate/refute that

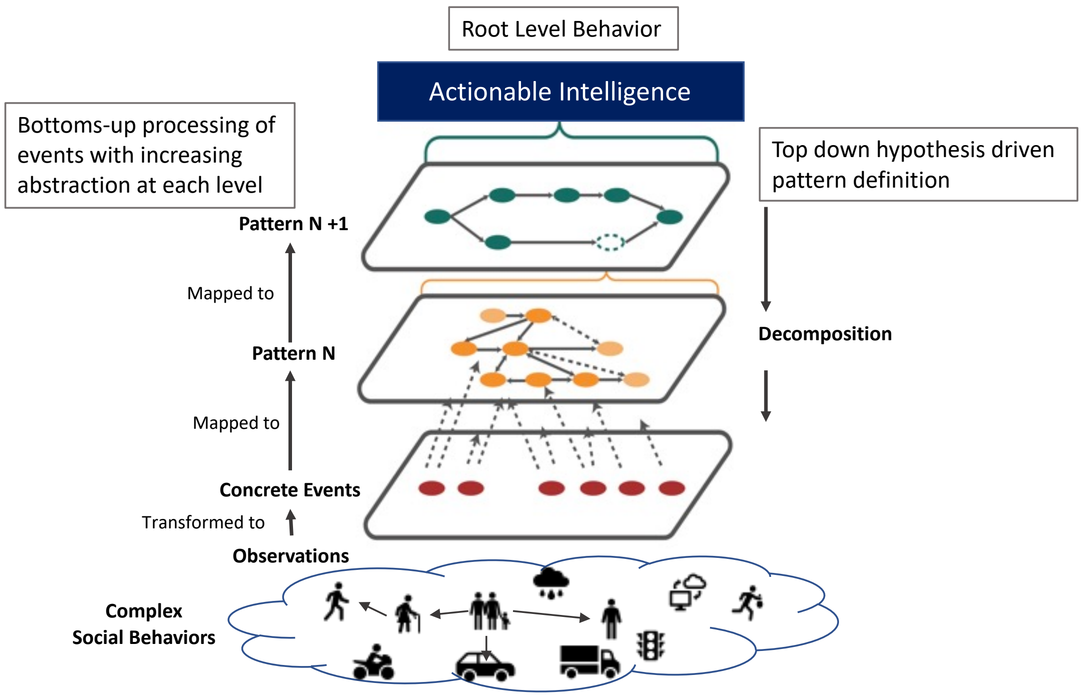
[2] https://en.wikipedia.org/wiki/Complex_event_processing

hypothesis, and the corresponding derived event patterns are matched by the observable events available to the system. This can be accomplished with no limit of hierarchical abstractions by the person who is conceiving these hypotheses.

Management of Complexity through HCEP

The notion of employing a hierarchical pattern matching approach is not really new or novel. The use of hierarchies and abstraction was formalized and has been put into wide practice since the 1960's with the emergence of systems engineering, and systems architecting in the development of complex systems, such as the Apollo space program and numerous other aerospace systems since then. In the early 2000's, Dr. David Luckham formalized the concepts of HCEP, patterns, and the use of abstraction hierarchies in his book *The Power of Events*[3]. This book informed many of our earliest design decisions.

The HCEP graphical concept is presented in Figure 2 and describes our HCEP solution. At the bottom of the figure, the diagram shows a seemingly complex social environment and a related assortment of behaviors and interactions. The goal from an analyst perspective is to derive insight from this activity.



[3] The Power of Events: An introduction to complex event processing in distributed enterprise systems/ David Luckham; 2002

Figure 2 Hierarchical Complex Event Processing Displaying Levels of Abstraction

The way to think about HCEP is both as a top-down hypothesis- driven pattern definition process, and a bottoms-up event matching and mapping process. One of the principle benefits of HCEP over the simple version of CEP is that it allows the analyst to address more difficult problems in an integrated context. For example, if we are looking at human behavior, there can be many factors that go into a profile, such as spending patterns, criminal violations, social media usage, personal associations and travel. Any one of these factors alone might have limited meaning, but in combination they could represent a potential risk. One of the most powerful features of HCEP is that it looks at non-events as well, meaning that a pattern can “look” for events that have not yet happened: in some instances, that non-occurence can be just as important as an occurring event itself. HCEP also continuously maintains the state of the hierarchy. If for some reason the system labeled a person’s profile as being high risk, and new data came in contravening that assessment, that person’s profile would be automatically adjusted.

Stated differently: Cogynt ESP continuously maintains the state of the event patterns for possibly millions of instances of hierarchies, and continuously maintains these hierarchies that are updated when new events arrive. At any time during the event pattern matching process, a notification event can be generated from the Cogynt ESP system to alert humans via a UI. This is a common feature within most decision support systems.

As discussed in the previous paragraphs, An HCEP solution must include three elements:

- *Computational Hierarchy* - A computational hierarchy is the arrangement of patterns that form a hierarchy, where patterns at lower levels feed higher level patterns, allowing an increase in abstraction and therefor understanding as patterns are populated up the hierarchy.

- *Stateful CEP* - Stateful CEP persists information about the state of ALL pending patterns allows for incremental evaluation of CEP patterns and to respond to new information (events) in the shortest time, while continuing to track potential patterns over a very long period of time. The persistence of this provides the means to “understand” the entire field of endeavor continuously.

- *A Domain Specific Language (DSL)* - that defines the logic of hierarchical patterns and the complex event processing. Our DSL is called Event Pattern Constraint Language (EPCL) and it is declarative and zero coding.

Cogynt ESP Platform as a System

Cogynt ESP is a highly robust and scalable ESP platform leveraging the best of breed open source technologies-Apache Kafka and Flink. Apache Kafka allows data to be ingested from most any available data source or format and streams the data [in the form of Kafka topics] to be processed within Apache Flink. Apache Flink provides a stream-based stateful compute engine for processing events and performing pattern matching.

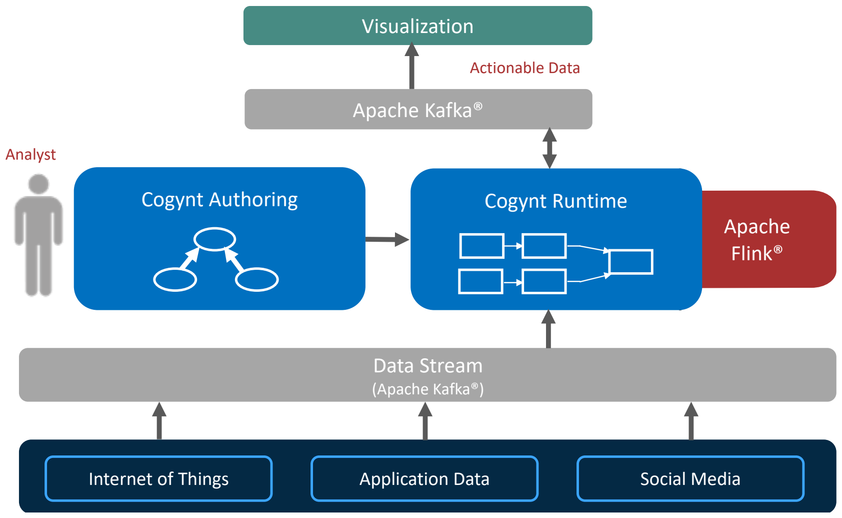


Figure 3 Cogynt ESP Platform

The Cogynt Authoring Tool is completely graphical, requires zero coding, provides the ability to define hierarchical patterns using the EPCL and maps the source data (Kafka topic schemas) to the patterns. Furthermore, the Cogynt Authoring Tool performs basic consistency checks prior to deployment to ensure the HCEP model is logically consistent and will execute once deployed. The Cogynt Runtime converts the deployed HCEP model into the Flink model representation, a Directed Acyclic Graph (DAG), from which Flink processes all events according to the HCEP pattern design. All HCEP generated events are published back into Kafka for reingestion into higher level patterns or can be published for visualization and further analysis.

The Wedding Example modelled in EPCL

In Figure 4, we show the visual concept of how a wedding would be implemented in EPCL.

These visual constructs are labeled representing all of the available constructs to define an event pattern. The variations of event patterns can be limitless based on the available event types, Elements and constraint logic that can be applied. The following bullets summarize the diagram shown in Figure 4:

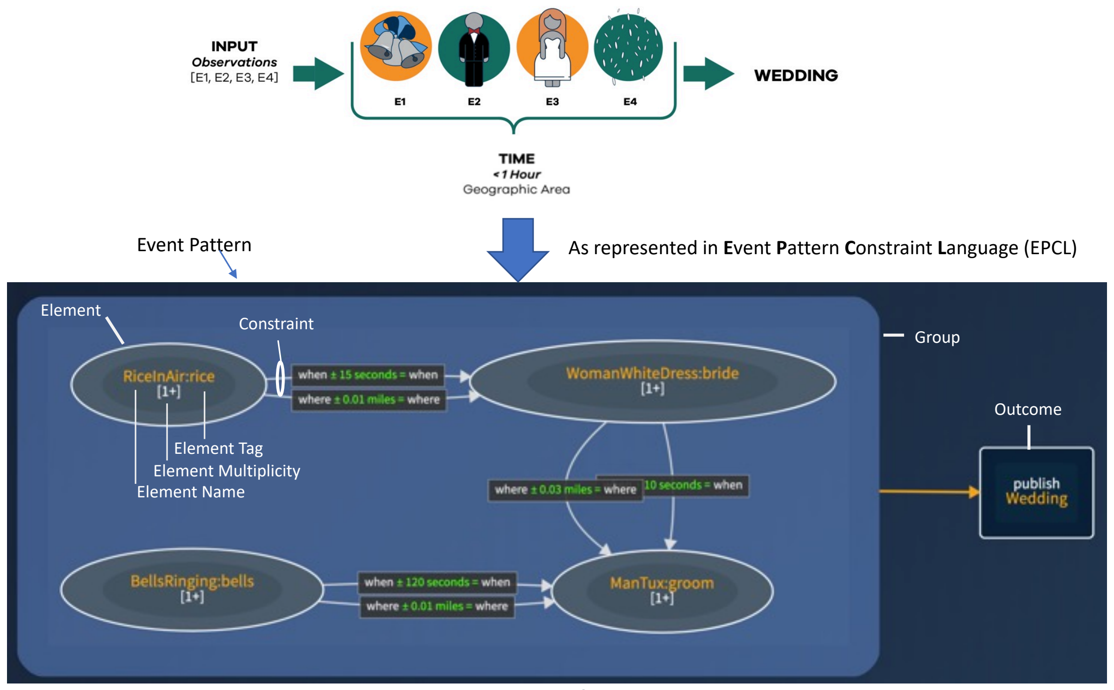


Figure 4 EPCL Pattern Definition – A Wedding

- <u>**Event Pattern**</u>: consists of all the graphical constructs shown in the diagram. This **Event Pattern** is defined by a total of 4 **Elements** (ovals) and 6 **Constraints** (arrows) encapsulated within a **Group** (enclosing rectangle). This **Event Pattern** (*A Western Wedding*) is fully satisfied if the defined **Elements** are matched with events, and the event data conforms to the **Constraint** operators. When the **Event Pattern is fully matched** an **Outcome** event is generated. Note: If no **Group** construct is used, partial pattern matches are allowed, and **Outcome** events can be generated directly from the **Element**.
- <u>**Event Type**</u> (not shown in diagram): is defined within the Cogynt Authoring Tool that consists of a *name* and a *data schema (set of data items and types)*. The event name and data schema can be discovered from a Kafka topic, or can be manually created.
- <u>**Element**</u>: Is represented as an oval construct that defines the **EventType** to be matched. Each **Element** is mapped to an **EventType** that specifies the *event name* and *schema* to be matched. An *Element Tag* is also provided to differentiate the use of an **Element** to maintain uniqueness if the same **EventType** is used multiple times in one pattern. The *Element Multiplicity* is an integer range that defines the number of events required, or allowed, to satisfy the **Element** within the pattern. If the lower and upper bound are the same only a single integer is displayed.
- <u>**Constraint**</u>: Is represented as an arrow construct. The direction of the arrow defines precedent of the **Constraint** operator for any comparison. An example **Constraint**

operator is "*greater than or equal to*" and "*less than or equal to*". There is currently a total of 7 comparison operators. (Note, a common confusion is to infer data flow from the arrow, when reading the diagram, but the arrow defines the direction to read the constraint between the two elements involved.)

- **Group**: is the rectangular construct that encompasses the **Elements** and **Constraints**. The **Group** construct ensures that all **Elements** and **Constraints** are satisfied before generating an **Outcome** event. A pattern may have multiple groups with disjoint or overlapping elements.
- **Outcome**: The rectangular object connected to one or more **Groups** represents the definition of a new event that can be mapped to higher level **Event Patterns** or used for notifications and visualization purposes. The **Outcome** allows for data to be propagated to higher levels within the abstraction hierarchy. When multiple inputs are directed to an **Outcome** the new event is created upon any group matching fully. In simple cases an **Outcome** can be linked directly to an element when only one element is required for the pattern.

A Wedding Reception - HCEP Example using EPCL

The purpose of this example is to illustrate all of the concepts discussed throughout this article using the Cogynt ESP Authoring Tool and EPCL. We start with Figure 5 which represents a pattern hierarchy with the lowest level patterns on the left and higher-level patterns to right, with arrows directly feeding the higher-level patterns. Note also, it's possible to have seemingly lower level pattern feeding directly a higher-level pattern several layers up. As stated earlier the conceptualization of patterns and abstraction layers starts with a hypothesis, which is a top down approach. In our hypothetical city, there are many types of weddings (different religions) occuring throughout the year and not all of the receptions are held at the same location as the wedding. We've also heard a rumor that inebriated wedding guests have been causing trouble and the police have been called in on a few occasions. The hypothesis is formulated in the form of a question. In this example, we are asking the following question? Considering all of the observable wedding receptions, how many of those receptions were purely happy occasions, and how many of the receptions experienced unruly situations that required a police call? This is the starting point for developing the event patterns.

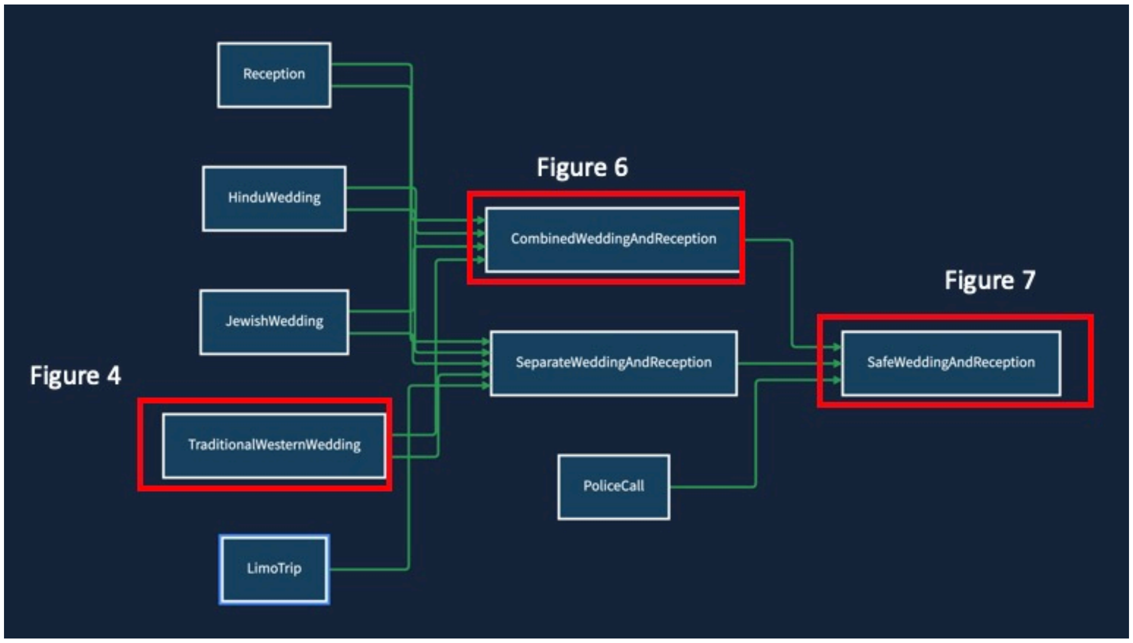


Figure 5 Cogynt Authoring Tool – Safe Wedding and Reception Pattern Hierarchy

Step one is to identify all weddings that occurred from the raw input observations. Jewish and Hindu weddings will have their own distinct patterns modelled. The outcomes of any of these base patterns will publish a "Wedding" **Outcome** event which will then be matched at the next higher-level pattern.

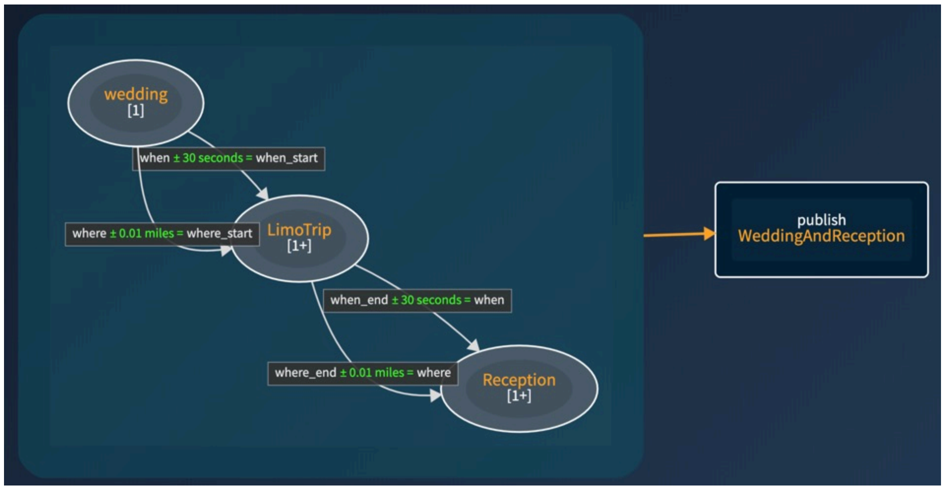

wedding
[1]
when ± 30 seconds = when_start
where ± 0.01 miles = where_start
LimoTrip
[1+]
when_end ± 30 seconds = when
where_end ± 0.01 miles = where
Reception
[1+]
publish
WeddingAndReception

Figure 6 CombinedWeddingAndReception Pattern

Step two is to use a higher level pattern to separate out *CombinedWeddingAndReception(s)* from *SeparateWeddingAndReception(s)*. The new Element and Constraint called *LimoTrip* is used for this because *LimoTrip(s)* are not needed for *CombinedWeddingAndReception(s)* located near, or at the same physical location. For this example, we look at the details of the *CombinedWeddingAndReception* Pattern shown in Figure 6. This Pattern shows 3 elements and

2 sets, or a total of 4 **Constraint**s. One set of **Constraint**s between the *Wedding* and *LimoTrip,* and the other set of **Constraint**s between the *LimoTrip* and the *Reception*. These **Constraint**s when combined define the allowable temporal and physical location proximities to the respective **Element**s, and if the **Event Pattern** is a match, then the Combined*WeddingAndReception* **Outcome** event is generated. This **Outcome** is then fed to the *SafeWeddingAndReception* Event Pattern defined in Figure 7.

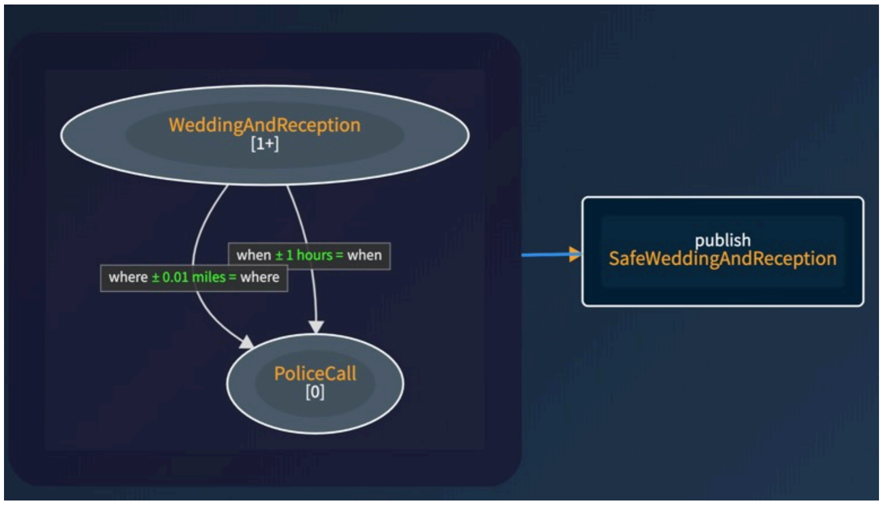


Figure 7 SafeWeddingAndReception Pattern

What is interesting about this **Event Pattern**, in Figure 7, **(the third step in this example)** is the *Police* **Element**. In this pattern we are looking for the absence of an event to match the pattern. If there is no observation of a police vehicle within 1 hour of the reception and within 0.01 miles, the **Outcome** *SafeWeddingAndReception* is generated. If the Police element is a match, then there will be no match of the pattern as a whole and the *SafeWeddingAndReception* will not be generated.

To recap, we defined 3 variations of a similar wedding patterns, each of which can generate a complex event (wedding). From there, there we determined, based on available location data, if the wedding and wedding reception was either in the same location, or held at a different location, and then we further determined based on the lack of presence of a police call (event) that the wedding reception was a safe and happy reception.

The last step in this process is the ability to deploy the HCEP pattern (model) into Apache Flink for execution. Figure 8 reflects the Flink job and corresponding DAG that is directly generated from the Cogynt ESP Authoring Tool. The reader should note that each box in the DAG is a software component that has to be written. The Cogynt Authoring Tool creates this code automatically upon deployment. The DAG that is shown is a deployment of the *Wedding Reception* example just described, which reflects how an analyst can create complex analytic designs and deploy them without having to do any programming.

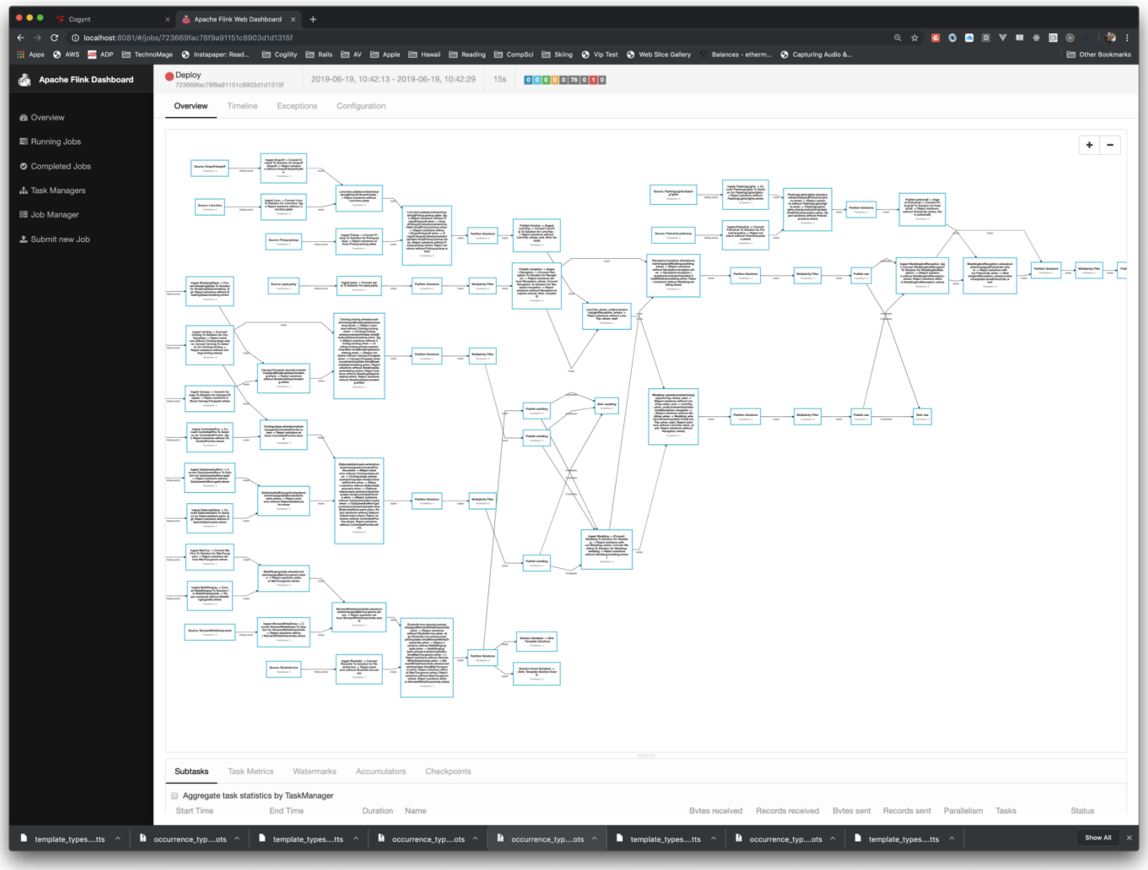


Figure 8 Apache Flink DAG View of the SafeWeddingAndReception HCEP Model

A Real World Application of HCEP

Our discovery of HCEP came about in 2009. We confronted a particularly challenging problem facing the U.S. Department of Defense (DoD) concerning Improvised Explosive Devices (IEDs). What the DoD needed was better intelligence in support of the forward deployed soldiers to better mitigate the IED risk. Our approach to this problem was similar to the wedding example described earlier. We defined IED patterns based on known bomb signatures. We treated the IED as a system made up of 5 basic components (as defined by IED experts), which became our basic event pattern, consisting of a *power source*, *switch*, *initiator*, *main charge*, and *container*. Based on this generic IED pattern, we modeled all the known signatures as individual HCEP patterns, for which there were several hundreds of variations. We also had access to an IED lexicon that defined all the IED technical terms (including synonyms) for each bomb type, so as to standardize the language across the community. Our job was to process all post-action reporting, including all IED related events, including follow-on forensic analysis of the devices that were encountered. Most of the reports consisted of unstructured text from which we extracted the terms that matched the predefined lexicon. We then conducted a pattern matching process and revealed to the soldier all the likely IED device types to be encountered in a given area. HCEP was also applied to evaluate the tactical and strategic motives behind the IEDs. This effort became a web-based application called the **J**oint **I**ED Analysi**S** **T**ool (JIST) which provided advanced situation awareness and decision support for one of the most difficult challenges faced in Operation Enduring Freedom.

Summary

Since 2009, Cogility has been a software analytic company that developed and applied Hierarchical Complex Event (HCEP) that analyzes both social and technical behaviors providing advanced situation awareness and decision support in the most challenging and time sensitive use cases. Coygnt ESP is our latest implementation of HCEP that employs a declarative, zero coding authoring capability that empowers analysts to work directly with the analytic without the need for programmers. This greatly enhances analyst agility in developing and changing their analytic designs serving as a force multiplier in terms of analyst effectiveness and efficiencies. Cogynt ESP is a scalable ESP platform that employs both Apache Kafka and Flink incorporating HCEP as its analytic core. This platform works with all event types including raw events generated from IoT devices, internet traffic and preprocessed data generated from AI and ML analytics, and provides actionable insights. Cogynt ESP can be a green field or a complement to existing technology and analytic investments by providing a packaged solution using Apache Kafka as the integration touch point. Cogynt ESP also provides advanced visualization options that includes highly integrated dashboards, analyst tools and risk assessment, which can be easily adapted to the client use case.