Real-time analytics for legacy data streams in health

Monitoring health data quality

Andrew Berry and Zoran Milosevic
Deontik Pty Ltd
Brisbane, Australia
andyb@deontik.com, zoran@deontik.com,

***Abstract*—*Healthcare organizations are increasingly using information technology to ensure patient safety, increase effectiveness and improve efficiency of healthcare delivery. While the use of health information technology (HIT) has realized many improvements, it has also introduced new failure modes arising from data quality and IT system usability issues. This paper presents an approach towards addressing these failure modes by applying real-time analytics to existing streams of clinical messages exchanged by HIT systems. We use complex event processing provided by the EventSwarm software framework to monitor data quality in such systems through intercepting messages and applying rules reflecting the syndromic surveillance model proposed in [4]. We believe this is the first work reporting on the real-time application of syndromic surveillance rules to legacy clinical data streams. Our design and implementation demonstrates the feasibility of this approach and highlights benefits obtained through improved operational quality of HIT systems, notably better patient safety, reduced risks in healthcare delivery and potentially reduced costs.*

Keywords-health; analytics; real-time; data quality; syndromic surveillance

I. INTRODUCTION

Healthcare organisations and facilities are increasingly using information technology to ensure patient safety, increase effectiveness and improve efficiency. While the use of health information technology (HIT) has realised many improvements, it has also introduced new failure modes arising from data quality and IT system usability issues [1]. Healthcare organisations and facilities are increasingly looking for solutions that can recognise data quality and usage issues, then act immediately and in real time. This automatic recognition of issues can improve quality and efficiency through faster reaction to opportunities and threats. The volume, complexity and velocity of data streams that feed such real time analytics are increasing. While database tools can provide the analytical capability and in some cases deal with the volume of data, they lack the ability to behave actively and respond in real time. Complex event processing technology (CEP) provides mechanisms that can address the real-time requirement (velocity), and with the right architecture, can also handle the volume and analytical capabilities required.

Health messaging is mature and quite entrenched in laboratories and other healthcare facilities. These facilities emit results, assessments and other messages typically structured in a form governed by the Health Level 7 (HL7) v2 standards [2]. The primary use of such data in current applications is for clinical (human) interpretation. The structured nature of this messaging, however, makes it feasible to attach a CEP engine to the stream of HL7 v2 messages received and emitted by a laboratory. CEP can then be used to monitor messages in real time and automatically detect data quality issues and clinical issues of interest. Such monitoring can improve the safety of the HIT systems and reduce the risk of harming patients [1].

In this paper, we describe the application of CEP techniques embodied in the EventSwarm framework [3] to legacy streams of HL7 v2 messages, using statistical techniques to identify unusual behaviours that indicate failure or data quality issues. A review of CEP capabilities and semantics is presented both to ensure readers are familiar with the technology and to position the distinguishing characteristics of EventSwarm. Upon recognition of such behaviours, an alert is sent to an operator or maintenance organisation to investigate and fix the problem as quickly as possible. The solution allows health organizations to continuously monitor the quality of data exchanged and thus the HIT systems from which the data originates - contributing to the improved safety of HIT components and ultimately to the safety of patients. The high cost and high demand on equipment used to produce laboratory messages also makes outages very costly. Thus there is also a considerable cost benefit in having such real-time analytics and alerting.

This paper is motivated by the recent work of Mei-Sing Ong et al [4] who proposed a new approach to early detection of HIT system failures using a syndromic surveillance method. They demonstrate the efficacy and feasibility of such surveillance against a static data set. We extend this work to show how the surveillance can be implemented in real time against existing data streams using the EventSwarm framework. Our solution also takes advantage of the HL7 data typing to extend the scope of the syndromic surveillance presented in [4].

The paper is structured as follows: section II identifies and discusses potential adverse impacts of HIT on patient safety and outlines the syndromic surveillance approach as a way of improving safety of HIT systems; section III provides a review of real-time analytics capabilities provided by CEP in general and EventSwarm in particular; section IV

describes the design and implementation of a solution using EventSwarm to implement syndromic surveillance against HL7 v2 data streams; section V discusses key features of the solution and other potential applications of CEP in monitoring HIT systems in general; and section VI concludes the paper.

II. Improving Patient Safety and Reducing Costs : use of IT for monitoring

A. Health IT and Patient Safety

A recent report published by Institute of Medicine [1] identifies a number of new concerns arising from the increased use of HIT in support of patient care. While the report highlights the benefits that HIT brings, including significantly improved quality of health care and reduced medical errors, the paper also provides some evidence of unintentional consequences of HIT on patient safety. It warns that if designed and applied inappropriately, HIT can add an additional layer of complexity to the already complex delivery of health care. This in turn can lead to unintended adverse consequences, for example dosing errors, failure to detect fatal illnesses and delayed treatment due to poor human–computer interactions or loss of data [1].

The concern for patient safety needs to be reflected in the design, implementation, use and maintenance phases of HIT systems. It is important to note that each of these stages needs to be informed by the enterprise or community context in which the HIT systems are to be implemented. This is reinforced in the report, stating that '*safety is an emergent property of a larger system that takes into account not just the software but also how it is used by clinicians. The larger system—often called a sociotechnical system—includes technology (e.g., software, hardware), people (e.g., clinicians, patients), processes (e.g., workflow), organization (e.g., capacity, decisions about how health IT is applied, incentives), and the external environment (e.g., regulations, public opinion)*' [1]. We take these considerations into account in the design and implementation of our solution, as will be discussed in section IV.

B. Health IT and Cost

HIT reduces the cost of healthcare delivery by reducing time spent by clinicians on time-consuming paper work and allowing them to focus on healthcare delivery. Costs are also reduced through automation of clinical workflows.

IT systems can also reduce costs arising from organizational or human factors, such as requesting duplicate lab orders or medications. These actions can be detected through the use of an appropriately configured monitoring infrastructure as will be discussed in section IV.

C. Syndromic Surveillance approach: Laboratory Orders and Results

In order to deal with early detection of HIT system failures, Mei-Sing Ong et al [4] investigated the feasibility of using a syndromic surveillance approach to detect health system failures. Syndromic surveillance is typically used in early detection of disease outbreaks and the authors wanted to determine whether a similar approach could detect HIT failures. They focused on detecting HIT failures in a laboratory information system (LIS) at a tertiary hospital, looking for anomalies in semantics (e.g. unexpected values) as well as structure (e.g. missing values) of data. A study of a one-year dataset from the hospital was performed for which the authors simulated four types of HIT failures and performed statistical analysis of the LIS data to detect those failures.

Their initial results were encouraging, suggesting that syndromic surveillance methods *'can be potentially applied to monitor HIT systems to facilitate the early detection of failures'* [4]. We leverage this work and show how elements of the syndromic surveillance system they described can be implemented through near real-time monitoring of HIT systems. In the following subsections, we briefly describe their approach, focusing in particular on the elements of relevance for our implementation.

Typical steps in undertaking syndromic surveillance are: defining syndromes; modeling baseline profiles; defining detection algorithms; and model validation. These steps are described next, based on the approach presented in [4], with a view of how they can be implemented using EventSwarm.

1) Defining syndromes

In the LIS scenario described, a number of symptoms or problems related to data quality were defined which are collectively referred to as the syndrome for LIS failures. These are: (1) loss of laboratory record created by a provider; (2) loss of data stored as a field within a laboratory record; (3) erroneous data being introduced into existing records (i.e. data entered by a provider differ from the data being stored or retrieved); and (4) unintended duplication of data (i.e. duplication of existing records not manually created by a provider).

In order to capture these four classes of failure symptoms, in their analysis the authors focused on the following indices and were looking for unexpected changes in them as signals for potential LIS problems:

1. Total laboratory records created in a given time frame so that an unexpected drop in the number of test records provides an indicator for data loss at the record level.
2. Total laboratory records with missing result so that an unexpected increase in the number of tests with missing results field provides an indicator for data loss at the field level.
3. Average test results for individual tests (serum potassium was selected as a proof of concept), so that an anomalous shift in the average level of serum potassium across all patients could signify that the data integrity of the LIS has been compromised.
4. Total number of tests of any types performed on the same patient within 24 hours of the same test being performed, so that a sudden increase in the number of duplicated tests requested is an indicator that test requests might be unintentionally duplicated.

The EventSwarm solution described in section IV focuses on items (3) and (4) above, since they are easily identifiable against an HL7 data stream. For (3) in particular, we are able to maintain statistics for all test results having numeric values.

2) Modeling baseline profiles

The purpose of this step is to establish a baseline against which new data can be compared in order to detect unexpected changes in indices. In [4], they have used two thirds of the dataset to establish the baseline values, and then used the remaining third to simulate new data arrival. The baseline values were established by applying statistical analysis to calculate mean, standard deviation and variance.

3) Defining detection algorithms

In this step, one defines rules for detecting significant unexpected changes. In [4], they have used a statistical control algorithm with the following triggers:

- If an observed data is 3 standard deviation (SD) above or below mean;
- Two out of three successive points more than 2 SD from the mean on the same side of the mean line;
- Four out of five successive points more than 1 SD from the mean on the same side of the mean line; and
- Six successive points on the same side of the mean line.

The EventSwarm solution implements each of these controls, evaluating the data in each HL7 message against each control.

4) Model validation

The purpose of this step is to perform validation of the model to detect failures. In the study this was achieved through simulating failures, each lasting for 24 hours with error rates ranging from 1% to 35%. The detailed description is beyond the scope of this paper, but the key conclusion was that syndromic surveillance methods were successful in identifying failures in HIT systems.

III. Real-time Analytics and EventSwarm

EventSwarm is a programming framework for complex event processing [5] or more recently referred to as real-time analytics. In this section, we provide an overview of CEP and EventSwarm in particular.

A. Origins

CEP is the analysis of streams of discrete information elements known as events. An event signifies an occurrence of interest, that is, it is a record of an observation made by a person or system. The *complex* adjective is used to highlight the application of correlation, aggregation and abstraction to raw event streams, allowing systems to recognize complex patterns of behaviour through analysis of the streams.

Luckham et al [5][7] and Bacon et al [6] pioneered research into CEP in the 1990s. Luckham focused on simulation and pattern matching, while Bacon et al focused on the construction of distributed applications using event-driven approaches. Rakotonirainy et al [8] published early work on building abstractions from event relationships in a similar timeframe. In [9], Berry extended and generalized this work and provided semantic foundations for describing event-based interactions in distributed systems. Early applications of CEP to real-time contract monitoring were explored by Milosevic et al [10] and Linington et al [11]. This body of work has steadily matured and is now embodied in a number of commercial products, including Apama [12], Esper [13], Infosphere Streams (IBM) [14], BusinessEvents (Tibco) [15], Oracle [16], SAP [17] and EventSwarm [3].

The features available in these products vary according to their targeted markets and applications, but many commonalities exist. Subsequent sections explore the typical capabilities of CEP systems and highlight the distinguishing characteristics of EventSwarm. A more extensive review can be found in [20].

B. Features

The core capability of all such systems and many preceding systems is to match the attributes of individual events, and as such, this capability is assumed. The key feature of CEP systems is the ability to recognize higher-level patterns through correlation, aggregation and abstraction. The following subsections describe some of these capabilities.

1) Event Expressions

Event expressions are the fundamental building block of a CEP system, allowing the specification of matching criteria for a single event. In the simplest cases, event expressions are based on static inspection of individual event attributes, for example, `e.color = red`. CEP systems can, however, offer far more sophisticated event matching based on statistical analysis, patterns and other computed abstractions.

In subsequent sections, event expressions are referenced by capital letters (e.g. `A, B` ...).

2) Filtering

Event streams associated with CEP are becoming increasingly voluminous, and a common mechanism used to constrain the scale of analytics is filtering. A filter applies an event expression `A` to a stream or streams and excludes events that do not satisfy `A` from any subsequent processing. Filtering event expressions can range in complexity from simple static inspection of event attributes to comparisons against complex computed abstractions, typically depending on the event expression capabilities offered by the system, as discussed in the preceding subsection.

The EventSwarm framework includes the ability to filter on any computed abstraction that matches a single event, including statistical analysis (e.g. attribute `e.x` is more than `r` standard deviations from the mean of all `e.x` attributes observed).

3) Pattern matching

Pattern matching is a core correlation mechanism and refers to the ability of the system to match event patterns using logical operators and sequencing over event expressions, including:

- AND, that is, a pattern `A AND B` matches pairs of events `e1,e2` such that `e1` satisfies `A` and `e2` satisfies `B`.
- OR, that is, a pattern `A OR B` matches one or more events `e1,e2` such that `e1` satisfies `A` or `e2` satisfies `B`.
- XOR, that is a pattern `A XOR B` matches individual events `e` such that either `e` satisfies `A` or `e` satisfies `B` but not both.
- repetition, that is, a pattern `A{n}` matches `n` occurrences of `A`. Repetition is sometimes considered a specialization of `AND`.
- sequence, that is a pattern `A then B` matches pairs of events `e1,e2` such that `e1` satisfies `A` and `e2` satisfies `b` and `e1` occurs before `e2` in time.
- causal sequence, that is, `A -> B` matches pairs of events `e1,e2` such that `e1` satisfies `A` and `e2` satisfies `B` and `e1` causally precedes `e2` using a comparison based on vector clocks [18].

In all cases above, these patterns are usually extensible to `n` event expression components. The availability of these pattern components varies across systems. While the first five above are assumed capabilities for a CEP system, the causal sequence capability is less widely available. The EventSwarm framework is capable of using causal precedence in sequence patterns, although as discussed in the following subsection, this has limited value outside closed, tightly coupled systems.

The above pattern types do not usually exclude the possibility of intervening events, that is, one or more events `e{n..m}` that occur between `e1` and `e2`. For most CEP systems, there is an underlying assumption of partial state, that is, a complete view of all behaviours is not available, making such exclusion inappropriate.

4) Complex events

A pattern match in a CEP system is sometimes called a *complex event*. A complex event is a set of events that indicates a behaviour of interest. Systems typically provide mechanisms to capture complex events in a higher-level structure that references the component events and the pattern or abstraction that it represents. EventSwarm uses the general term *Activity* for this type of event, and the specific term *ComplexExpressionMatch* for events generated as a result of matching a pattern.

5) Time and ordering

As implied in the previous section, there are a variety of ways to manage the ordering of events in time. A key difficulty in CEP is that different event sources also have different time sources. A further complication arises because the latency of delivery is such that events are often delivered out of order. Thus a CEP solution needs to have a considered and careful approach to the use of event timestamps and event ordering. This approach is particularly important in the handling of sequence patterns.

Some systems will use the clock time on the system that is processing the events. This is simple and ensures that events can be presented for processing in a time-consistent order. It also has many deficiencies, in particular, it makes sequencing of events particularly inaccurate due to latency and clock skew when distributed event sources are used. In such systems, sequence patterns must be treated with considerable caution.

Other systems will use the event timestamp of events and provide mechanisms to handle out-of-order events. This is more complex to implement and requires that application design considers event sources and timing to ensure that out-of-order events are correctly handled. When combined with appropriate buffering of events in expressions, it provides a much more robust and reliable result. This approach also allows for flexible replay in simulation and testing scenarios, ensuring time windows and other time-driven expressions are correctly evaluated. This approach still suffers from inaccuracy arising from clock skew across the event sources. Providing an allowance for clock skew in time comparisons can assist, that is, treating events from distinguished sources as concurrent if their timestamps differ by less than the maximum anticipated clock skew. It should be noted that this skew allowance can result in false negatives when matching, so must be applied judiciously. Due to the difficulties associated with sequencing, it is often better to use a logical `AND` within a time window instead of a sequence for patterns where matched events are close in time.

It is also feasible in some cases to implement ordering comparisons based on vector clocks [18] to define a causal order. Few CEP systems provide this capability, in part due to the relative difficulty of implementation: establishing causal precedence is generally only possible in a closed system with cooperating components and limited communication mechanisms.

The EventSwarm framework uses event timestamps and buffering to provide a robust and flexible time and ordering implementation. It uses *before*, *after* and *indistinguishable* or *parallel* relationships between events to give flexibility in ordering. Time skew allowance and causal ordering based on vector clocks are available, although as noted, it is difficult to use causal ordering except in closed environments. These time handling features are quite novel in an implementation of complex event processing, giving EventSwarm considerable flexibility in dealing with multiple, independent data sources.

6) Sliding Time Windows

Sliding time windows are an aggregation mechanism that limits the scope of correlation and abstraction to a window that moves relative to the current time. A sliding time window includes all events whose timestamp is within a defined period before the current time.

An important aspect of sliding time windows is in how the *current time* is determined. Some systems will use the clock time; others will use event timestamps to determine the window bounds, that is, an event `e1` remains within the window until a subsequent event `e_n` is observed with a timestamp more than the window period ahead of `e1`. This latter approach is more robust in a distributed context, although there are still window accuracy implications for both approaches when events are received out-of-order or

with high latency. The EventSwarm framework uses the event timestamp approach.

7) Statistical Analysis

Statistical calculations are essential in many applications of CEP. A CEP framework typically provides the ability to calculate statistics on attributes of events, or in some cases, on other values computed during processing. These calculations can then be used to detect, for example, when the average price of a stock has risen by more than 10% or when a health care laboratory result falls more than 3 standard deviations outside the mean value for that result. This provides a particularly powerful way to detect unusual or important events.

Combining statistical analysis with sliding time windows can further extend the power of a CEP framework, for example, allowing us to compare the long-term average traded volume of a stock with the average traded volume in the last hour. EventSwarm provides the ability to calculate statistics on sliding time windows and use the statistics in expressions.

8) Language, Expressiveness and Complexity

There are two major approaches towards defining event patterns and abstractions in CEP: on one side there are approaches using domain specific languages, often based on SQL (e.g. Esper [13]); on the other side there are approaches based on general purpose programming languages. Both have strengths and weaknesses, notably:

- Domain specific languages can provide a more approachable learning path and hide complexity in the query processor and optimizer; programming frameworks typically require a deeper understanding of the concepts and implementation, thus a longer learning curve.
- Hiding complexity in a domain specific language typically reduces flexibility and expressiveness (e.g. distribution, parallelism as discussed below); programming frameworks based on a general purpose language give access to all facilities available in that language, thus maximizing flexibility and expressiveness.
- Lack of standardization across domain specific languages means that experts are uncommon and users rely on the CEP vendor to resource projects; programmers are readily available to resource projects when a general purpose programming language is used.

Note that in all cases, there is potential to define patterns or abstractions that quickly exhaust the available resources due to combinatorial explosion. There is also no guarantee that expressions correctly match the behaviours required for the underlying business need. Thus a domain specific language does not absolve users of the need for a software development lifecycle including requirements gathering, implementation, testing and deployment, but it can reduce the cost of the implementation phase for simple patterns. This implementation advantage is eroded as complexity increases.

EventSwarm provides a programming framework based on the Java programming language.

9) Distribution and Scalability

The increasing volumes of raw data being generated by systems today require distribution and horizontal scalability to effectively apply real-time analytics. A common term now used for this voluminous data is *big data in motion*. CEP systems differ considerably in their approach to distribution and scalability, and the approach often depends on the choice of a domain specific language versus a programming framework, as discussed previously. The relative scalability is characterized as follows:

- Domain specific languages typically have system-determined distribution and parallelism. Such languages often imply a degree of shared state, limiting the ability to distribute pattern execution. Coarse-grained distribution is typically handled through manual deployment of separate instances.
- Programming language component-based approaches provide pattern and abstraction components that can be distributed according to their semantic constraints and the business need, and then connected using raw or abstracted event feeds. This is typically more flexible than domain specific languages for distribution and parallelism, but the construction of applications can be more complex and error-prone.
- Event-driven approaches extend component-based approaches by requiring that the behaviour of each component is a function of the input events. These approaches can allow arbitrary distribution and massive scalability because behaviour is consistent in the face of both distribution and parallelism.

EventSwarm uses an event-driven approach, where distribution capability is implicit in the semantics. It also provides some useful, semantically-consistent abstractions for parallelization

In our experience, a key scalability dimension for all implementations is memory usage. In-memory processing is required for low-latency evaluation of patterns and abstractions but this limits the capacity of a processing node. Storing events on disk provides greater scalability, but increases latency and complexity. Thus there is a trade-off between in-memory and disk-backed event processing. EventSwarm is focused on in-memory processing, with filtering and distribution capabilities used to achieve the necessary scalability.

10) Near-real-time capability

The majority of CEP systems evaluate expressions, patterns and other abstractions continuously as events arrive at the point of processing. As such, they may be considered to have *near real-time* capability. Some CEP solutions rely on storing events in an in-memory database and periodic evaluation of queries. This caching and periodic evaluation increases latency and typically increases memory and computing capacity requirements for the software.

The EventSwarm framework operates continuously as events arrive, and thus can be considered near real-time.

C. *EventSwarm Architecture and Usage*

As discussed in preceding sections, EventSwarm implements a near-real-time, event-driven approach to CEP. It provides a range of predefined abstractions and pattern components implemented in the Java programming language. There are two typical styles of application built using this framework:

1. Applications built for specific, pre-defined patterns or abstractions
2. Domain-specific applications that allow end users to define new patterns

In both cases, the following activities are required in building the application:

1. Identifying types of events and their sources
2. Implementing *channels* to collect events from previously unimplemented sources
3. Implementing actions required when a pattern or abstraction is matched

For applications built to match specific, pre-defined patterns, the necessary patterns and abstractions are coded in the application and tested statically. For applications that allow users to define domain-specific patterns, the following additional steps are required:

1. Identifying the types of patterns required to meet business needs
2. Implementing a user interface that allows the user to safely and conveniently define new patterns of the identified types and select relevant actions
3. Implementing mechanisms to generate and deploy new pattern instances with associated actions

As discussed in the preceding subsections, the usual software development lifecycle phases apply for both pre-defined and user-defined patterns.

For applications that allow definition of new patterns, we typically use Ruby and its JRuby runtime to build the user interface and pattern instance configurations. In doing so, we couple the performance of Java execution for event processing with the flexibility of Ruby for the more dynamic, user-oriented tasks.

The design of EventSwarm is intended to support lean, agile and focused solution development. The application described in this paper, for example, was built to proof-of-concept state in two weeks, including tools for generation of test data and an optimized HL7 parser.

IV. Applying EventSwarm to Orders and Results

In this section, we provide an introduction to the technical details of HL7 v2 messaging and describe how EventSwarm is configured to monitor the data quality and cost metrics described in section II.

A. *HL7 v2 Messaging*

The healthcare industry in the USA, Australia and many other countries predominantly uses HL7 version 2 (v2) messages to transmit laboratory orders, results and other clinical information. For example in Australia, over 90% of pathology reports were delivered electronically as HL7 v2 messages in 2012. Individual messages are transmitted in a "fire and forget" manner with distinguished acknowledgement messages to confirm delivery and acceptance. HL7 v2 was originally designed to operate over an OSI 7-layer network architecture but messages are more commonly transmitted using TCP/IP and associated protocols in current implementations.

The HL7 v2 standards define a large number of different message types [2], each identified by a 3-alphabetic-character prefix to indicate a functional type (e.g. order message), followed by a caret and a more specific content type. For example, `ORM^O01` identifies a general order message and `ORU^R01` identifies an unsolicited observation result. The content of each message is organized into segments, with each segment also having a type identifier, for example, an `OBX` segment contains a clinical observation or result. All messages have an `MSH` segment containing message header information. Within each segment there can be data-elements with fields and sub-fields. Segments can also have nested segments. Values within segments are typed, including numeric, coded text and free text. Coded text fields are taken from a well-defined namespace (i.e. a pre-defined enumeration of text tokens). The example depicted in Figure 1. shows a sample OBX segment and describes its components:

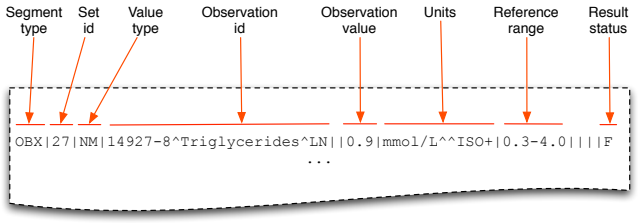

Figure 1. HL7 v2 OBX Segment]

Thus extracting data from an HL7 message involves parsing the message, finding the segment or segments that contain the required data, and extracting the value from the field.

There are a few key issues associated with processing HL7 v2 laboratory orders and messages, specifically:

- Segments may by typed `FT` (free text) and thus intended for human interpretation. It is difficult for an analytics engine like EventSwarm to deterministically evaluate these free text fields, although work reported in [22] shows promise.
- Coding systems and value types vary widely, often differing across laboratories. Normalization is sometimes possible but not feasible in the general case, thus processing rules might need to be specific to a laboratory.
- De-jure and de-facto content standards with normative coding systems exist but are not always adhered to, with systems relying on the knowledge of clinicians to interpret results. Such content standards are typically country-specific.
- Coded values sometimes allow extension, that is, the value can be selected from an enumeration or a custom value can be used. Such custom values

introduce complexity and non-determinism into rules, especially where such rules are used across health jurisdictions or organizations.

To a great extent, the issues above arise because HL7 v2 message content was primarily intended for human consumption first and machine processing second. The subsequent sections describe the design of a monitoring solution that takes these issues into account.

It is worth noting that the HL7 v2 format is relatively compact. HL7 version 3 and the associated Clinical Document Architecture (CDA) standards use XML as the base syntax. The advantages are improved structuring and off-the-shelf parsing and validation tools, but this comes at the expense of compactness and processing overheads. For real-time analytics at the scale required for our operational context, the compactness and lower processing overhead of HL7 v2 is a significant advantage.

B. *Operational Context*

The assumed operational context for this solution is within a large integrated healthcare delivery organisation providing healthcare services for its members. The organisation is national in scope but with regional operations. The focus of the implementation is on pathology orders. The scale of the organisation in terms of laboratory orders and results is characterized by the following metrics:

- 1,000,000 pathology orders per day
- approximately 50% of pathology orders have distinguished test identifiers (i.e. not free text descriptions)
- approximately 2% of pathology orders contain duplicate tests
- Each laboratory processes and average of 50,000 orders per day
- Current average detection time for laboratory data quality issues is 1 day

It is worthwhile to consider the scale of this solution, and in particular, the memory requirements associated with duplicate order detection. For an in-memory implementation and assuming a 30-day duplicate detection window, non-distributed approach would need to hold 30,000,000 records in memory. At 1KB per order, that equates to 30GB of memory. The processing and deployment architectures described in subsequent sections aim to reduce this memory footprint as much as possible. In most cases, a single instance of EventSwarm on a modern multi-core CPU will be able to process this volume of messages and the associated expressions with capacity to spare. Network and IO bandwidth considerations could pose problems for a centralized solution, suggesting a regional solution might be appropriate. Further discussion relating to regional deployment is presented in the following section.

C. *Design Overview*

Our solution operates as follows:

1. Extract a copy of each message transmitted to and from a laboratory through intercepting the message in the middleware infrastructure, or through having the source system send a copy of the message to an EventSwarm processing node;
2. Feed each message into a configuration of EventSwarm processing components that implement rules for data quality and duplicate orders; and
3. Send an alert to the laboratory operator when a message matches an alerting rule.

This high-level design is depicted in Figure 2.

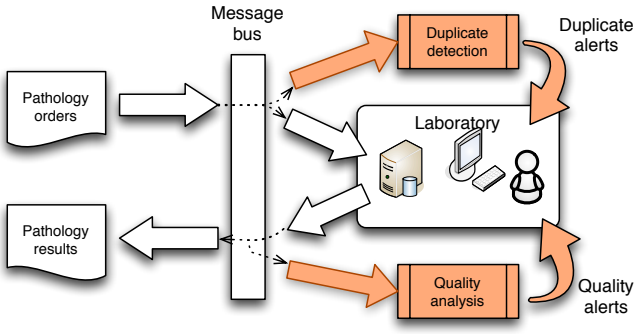

Figure 2. Functional architecture

The data quality metrics require pre-loading of historical data to establish a stable statistical profile for messaging, thus our EventSwarm solution also provides a mechanism to pre-load historical data. It allows historical data to be pushed through the solution to establish a baseline prior to connection to a live data feed. This data is readily available in laboratory environments due to healthcare archiving requirements. The baseline is extended as new messages are processed, providing increased statistical stability over time.

EventSwarm processing nodes are deployed either regionally or per laboratory to ensure scalability and low latency. This deployment reflects the usual manner of health-care delivery, that is, a patient will typically receive all of their health care services within their home region, and will typically have laboratory tests of a particular type processed at a single laboratory site. Localization also minimizes the variation arising from differences in content types and coding systems, for example, issues associated with coding extensions discussed in A. Further, localized deployment provides basis for incremental deployment and evaluation of benefits before progressing to larger scale deployments. This deployment architecture is depicted in Figure 3.

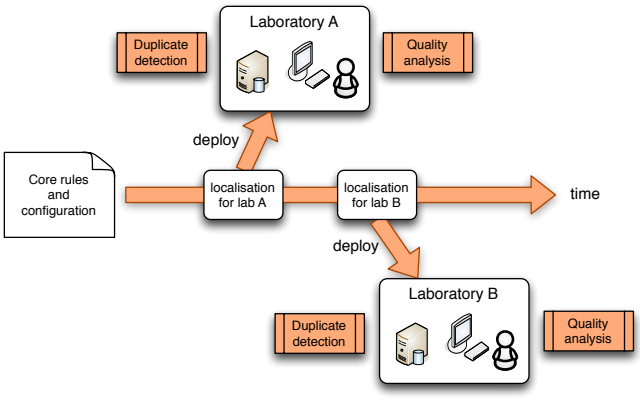

Figure 3. Deployment architecture

D. *Message interception*

In the operational context, messages are transmitted to and from the laboratory using message-oriented middleware. EventSwarm can connect directly to the middleware and receive copies of all messages as they flow through the organisation.

E. *Data Quality Monitoring*

The data quality monitoring configuration is intended to detect instances where numeric observations match the rules identified in section II, that is:

1. A numeric observation is more than 3 standard deviations from the mean;
2. At least 2 of the last 3 numeric observations are more than 2 standard deviations from the mean;
3. At least 4 of the last 5 numeric observations are more than 1 standard deviation from the mean; and
4. The last 6 numeric observations are all on the same side of the mean.

In contrast to [4], we are able to apply these rules to all numeric observations in a message rather than a single observation, grouping them by the observation type id in the `OBX` segment containing the observation. The configuration of components is depicted in Figure 4. This configuration is attached to a stream of laboratory reports.

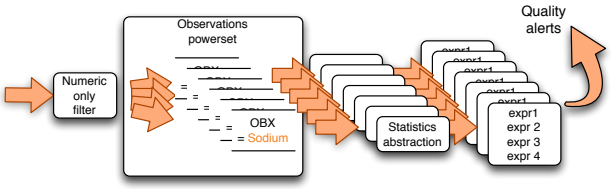

Figure 4. Data quality monitoring configuration

The configuration of components is described as follows, in order of processing:

1. We first apply a filter to exclude reports that do not contain any numeric observations.
2. We then split the reports using a powerset to maintain a subset for each numeric observation type id. If a report has multiple numeric observations, then it will be added to the subset for each of those types (i.e. the subsets intersect).
3. A statistics abstraction is attached to each subset, maintaining statistics for all observations of that type.
4. Four monitoring expressions are also attached to the subset, with references to the historical mean and standard deviation maintained by the statistics abstraction:
    a. rule 1 above is directly evaluated by comparing the current observation with the historical mean, firing if the observation value is more than 3 standard deviations from the mean.
    b. rules 2 through 4 above use a LastN window to capture the last N observations, and a MatchCount expression to detect when more than M observations satisfy the statistical expression. For example, rule 3 would use a LastN window with a size of 5 and a MatchCount expression that fires when 4 of the collected observations have values greater than 1 standard deviation from the mean.
5. For each monitoring configuration, two actions are connected:
    a. An action to notify the laboratory operator when a match occurs, including a URL for a page that can display the matching message(s)
    b. An action to store the matching message(s) on disk for subsequent audit and investigation

The majority of processing components used in this configuration are off-the-shelf components from the EventSwarm framework. Components that were added specifically for this configuration were:

1. Components to parse HL7 messages and extract data from those messages. These components are re-usable in any other EventSwarm application using HL7 messages.
2. Components to render matching HL7 messages for display to the laboratory operator.
3. A custom action component to integrate with the proprietary notification system used in the operational context.

Note that this configuration has predictable memory requirements, since we only need to maintain the last 6 observations of each type in memory.

Initial and subsequent loading of historical data is achieved by establishing the configuration without any monitoring expressions or actions configured, then processing the historical documents. Once historical data is loaded, the monitoring expressions and actions are added. If required for speed of restarts, a future extension might persist the statistical abstraction.

F. *Duplicate Order Monitoring*

The duplicate order monitoring component is required to detect duplicate laboratory orders for a patient in a defined time window. As discussed in section IV-B, the scale of duplicate detection is significant so we have explored the business problem to identify ways that the scale can be contained without sacrificing any business benefits. There are a number of key characteristics of the rules that allow us to contain the scale:

1. The useful lifetime of the test results varies for each test. For example, a full blood count has a useful life of about 30 days, thus duplicates should be detected in a 30-day window. In contrast an INR (blood clotting time) has a useful life of about 4 days, thus duplicates should be detected in a 4 day window. Shorter windows can significantly reduce memory requirements.
2. Some tests are inexpensive and unobtrusive for the patient. Duplicate detection on such tests has limited benefits. Other tests are particularly expensive or

intrusive, so focusing on such tests can deliver maximum value from the monitoring.

3. It is inappropriate for the system to reject duplicate orders. Orders will sometimes have associated free text to indicate why a duplicate has been ordered, and in some cases the previous test results might not be accessible to the requesting clinician (e.g. for privacy or other reasons). Thus an operator will have to intervene to determine an appropriate rectification. This increases the cost of rectification, particularly if the ordering clinician must be consulted. The cost of rectification needs to be weighed against the cost and intrusiveness of the test.

We are able to configure duplicate detection using the above business considerations to reduce memory requirements. In particular, we use a "white-list" of test types to identify tests for which the benefit of duplicate detection outweighs the cost. The use of regional processing nodes also allows us to reduce the scale of any single processing node. The configuration of components is depicted in Figure 5.

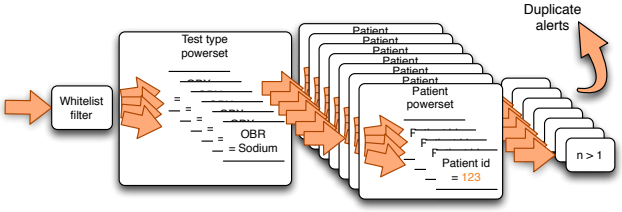

Figure 5. Duplicate order monitoring configuration

This configuration of components is described as follows, in order of processing.

1. Orders are first filtered against a configurable white-list of tests. If no listed tests are present in the order, the order is ignored.
2. Orders are split using a powerset to maintain a subset for each test type. The useful life of each white-listed test is also maintained, and the subsets of the powerset are configured to use a sliding time window with a size equivalent to the useful life of the test. Note that since an order might contain multiple tests, that order can appear in multiple subsets.
3. Orders for each test are split again using a powerset to maintain a subset for each patient using the patient identifier information in the order.
4. For each patient subset, a threshold monitor is attached, firing any registered actions when the size of the set is greater than one
5. Two actions are connected to each threshold monitor:
    a. An action to notify the laboratory operator when a duplicate is detected, including a URL for a page that can display the matching message(s)
    b. An action to save the orders for future audit and investigation

It is instructive to note that a sliding time window propagates event removals through the downstream configuration, meaning that test orders older than the useful life of the test (i.e. outside the time window) are automatically removed from any downstream sets.

As with the configuration for data quality monitoring, the majority of this processing is performed using off-the-shelf components from the EventSwarm framework. The new HL7, rendering and action components described in the preceding section are re-used here, leaving only two new components for this monitoring:

1. A white list matching component. This is a simple extension of an existing class in EventSwarm.
2. A set factory component to create test-specific sliding time windows. This is used by the powerset to create subsets test type.

The memory usage of this configuration on a per-laboratory basis is estimated as follows:

$$50000 * 50\% * \texttt{white list}\% * \texttt{mean life} * \texttt{size}$$

That is, number of daily events reduced by 50% to exclude unstructured orders, and reduced by the percentage of orders that match the white list, multiplied by the mean life in days of the white-listed tests. If we assume a white-listed volume of 30%, a mean life of 10 days, and an average order size of 1KB, we get:

$$50{,}000 * 0.5 * 0.3 * 10 * 1\text{KB} = 75{,}000\text{KB or } 75\text{MB}$$

Thus our configuration is easily capable of residing in-memory on a single processing node.

V. Discussion and Future Work

In this paper we show how a CEP approach can be used to implement real-time analytics against existing healthcare messaging and improve HIT safety and efficiency. This is achieved through monitoring statistical quality metrics against pathology laboratory reports and through detection of duplicates in pathology orders. Alerts are used to notify necessary parties when quality issues or duplicates are detected.

We have implemented this solution using test data derived from messages used for conformance testing against Australian HL7 v2 pathology messaging standards. The solution has been demonstrated to a selected set of stakeholders. As suggested by our analysis, the memory requirements of the solution are well-contained. The duplicate detection and data quality solutions are able to process 2000-3000 HL7 v2 messages per second each on a single CPU core, which is more than adequate for likely deployment scenarios.

This experience provides a basis for the implementation of other, similar applications based on monitoring of HL7 v2 messages, including those that involve natural language description of medical terms as reported for example in [22] in which they generate structured documents for cancer reports from natural language observations. In addition,

EventSwarm could be also used to detect network problems or problems with laboratory equipment or other medical devices if they support access points for operational quality checking.

In general, CEP can be used in many other eHealth applications, for example: *i)* real-time monitoring of patient conditions in intensive care units, where similar surveillance of patient condition indicators could be used to detect critical changes in patient condition; *ii)* detection of disease outbreaks through monitoring a combination of streams including social networking, clinical information systems and messaging to registries of reportable diseases; *iii)* detecting fraud in e-Medication systems such as attempts to request multiple doses of the same drug; and *iv)* exchange of alerts between health providers and emergency services organizations to assist in crisis management scenarios.

CEP can be also used in detecting non-clinical causes of harm to patients such as those arising from suspicious cybercrime activities as indicated in [4] or possible violations of privacy policies. Finally, CEP can be used to facilitate better care coordination interactions, for example: *i)* laboratory alerting systems that apply clinical decision support to identify critical results and page physicians when they occur [19]; *ii)* the detection of delays in healthcare delivery as specified in the recently proposed care coordination service [21]; and *iii)* to remind patients to take medication or make an appointment associated with a referral.

We intend to further experiment with other uses of EventSwarm in healthcare, both to improve safety of HIT systems and also to support other monitoring and alerting applications in eHealth as discussed above.

VI. Conclusions

This paper describes the architecture and implementation of a real-time analytics solution to support improving quality and safety of HIT systems and reducing unnecessary costs due to inadvertent issue of duplicate lab orders. The solution applies syndromic surveillance approach over HL7 v2 messages received and transmitted by clinical laboratory systems. The paper extends work of Ong et al [4] who demonstrated the feasibility of syndromic surveillance algorithm for monitoring of HIT systems. The novelty of our approach is in designing and implementing syndromic surveillance rules in a large and complex healthcare environment with minimal disruption to existing HIT environments through interception and analysis of HL7 v2 messages.

We believe this is the first reported application of a CEP technology for monitoring of quality and safety in HIT systems, and should pave the way for ongoing improvements in HIT system quality and safety through real-time analytics.

The paper also presents a review of capabilities and semantics of CEP technologies, highlighting the flexibility, utility and novel features of the EventSwarm CEP framework. While implementations of this technology are relatively mature, there are many factors that influence the ability to provide near-real-time analytics for big data in motion. The results presented in this paper demonstrate the ability of EventSwarm to provide such solutions.

VII. Acknowledgments

We would like to thank Prof Enrico Coeira for initial review of this paper. We would also like to thank Klaus Veil and Grahame Grieve for planting the seeds that gave rise to this work, and to Dr Michael Legg for fruitful discussions around pathology practices and technology.

References

[1] Committee on Patient Safety and Health Information Technology; Institute of Medicine. Health IT and patient safety: building safer systems for better care. The National Academic Press, 2012.

[2] HL7 Standard Version 2.4: An Application Protocol for Electronic Data Exchange in Healthcare Environments. Health Level 7, Ann Arbor MI USA, http://www.HL7.org.

[3] http://www.deontik.com/Products/EventSwarm.html

[4] Mei-Sing Ong, Farah Magrabi and Enrico Coiera**, "**Syndromic surveillance for health information system failures: a feasibility study, " *J Am Med Inform Assoc* published online November 26, 2012

[5] Luckham, D., The Power of Events, An intoduction to Complex Event Processing in Distributed Enterprise Systems, Addison Wesley, 2002

[6] Jean Bacon, Ken Moody, John Bates, Richard Hayton, Chaoying Ma, Andrew McNeil, Oliver Seidel, Mark Spiteri, Generic Support for Distributed Applications" IEEE Computer, March 2000, pp 68-76

[7] http://complexevents.com/stanford/rapide/

[8] A. Rakotonirainy, A. Berry, S. Crawley, and Z. Milosevic: Describing Open Distributed Systems: A Foundation. The Computer Journal, 1997

[9] Andrew Berry: Describing and Supporting Complex Interactions in Distributed Systems. The University of Queensland, July 2001

[10] Zoran Milosevic, Peter Linington, Simon Gibson, James Cole, Sachin Kulkarni, On design and implementation of a contract monitoring facility, IEEE Conference on Electronic Commerce, the 1st IEEE Workshop on Electronic Contracting (WEC04)

[11] Peter F. Linington, Zoran Milosevic, James B. Cole, Simon Gibson, Sachin Kulkarni, Stephen W. Neal: A unified behavioural model and a contract language for extended enterprise. Data Knowledge Engineering 51(1): 5-29 (2004)

[12] http://www.progress.com/en-au/apama/index.html

[13] http://esper.codehaus.org/

[14] http://www-01.ibm.com/software/au/data/infosphere/

[15] http://www.tibco.com.au/products/event-processing/complex-event-processing/default.jsp

[16] http://www.oracle.com/technetwork/middleware/complex-event-processing/overview/index.html

[17] http://www54.sap.com/solutions/tech/database/software/sybase-complex-event-processing/index.html

[18] C. Fidge. Logical time in distributed computing systems. IEEE Computer, pages 28–33, Aug. 1991.

[19] HL7 Version 3 Standard: Decision Support Service (DSS), Release 1, August 2011

[20] Gianpaolo Cugola, Alessandro MargaraProcessing. Flows of Information: From Data Stream to Complex Event Processing, ACM Computing Surveys (CSUR), Volume 44, Issue 3, June 2012

[21] http://wiki.hl7.org/index.php?title=Care_Coordination_Service

[22] Anthony Nguyen, Julie Moore, Derek Ireland, Guido Zuccon, Deanne Vickers, Bevan Koopman, Michael Lawley, Shoni Colquist, Streaming medical report analytics at increasingly "big data" scale, in Australian HISA big-data conference, 2013