Java is a trademark of Sun Microsystems, Inc.

JavaOne℠

Applying CEP with a Stateful Rules Engine for Real-Time Intelligence

Adam Mollenkopf
FedEx Custom Critical
Strategic Technologist

Mark Proctor
Red Hat
Drools Lead

Applying CEP
Agenda

- **Intro to Complex Event Processing (CEP)**
- **Stateful Rules Engine + CEP = Real-Time Intelligence**
- **CEP Applied – FedEx Custom Critical Case Studies**
  - Demonstration, Architecture Review, and Code Walk-Through

Complex Event Processing (CEP)

- "**Complex Event Processing**, or **CEP**, is primarily an event processing concept that deals with the task of processing multiple events with the goal of **identifying** the **meaningful** events within the event cloud.

- CEP employs techniques such as **detection** of complex **patterns** of many events, event correlation and abstraction, event hierarchies, and **relationships** between events such as **causality**, **membership**, and **timing**, and event-driven processes."

-- wikipedia

Typical Event Processing Scenarios

- Usually receive large volume of events, but only a small percentage are of **real interest**.
- Individual Events are usually not important. The system is typically more concerned about **patterns** of **related events**, their **relationships**, and what can be **inferred** from them.

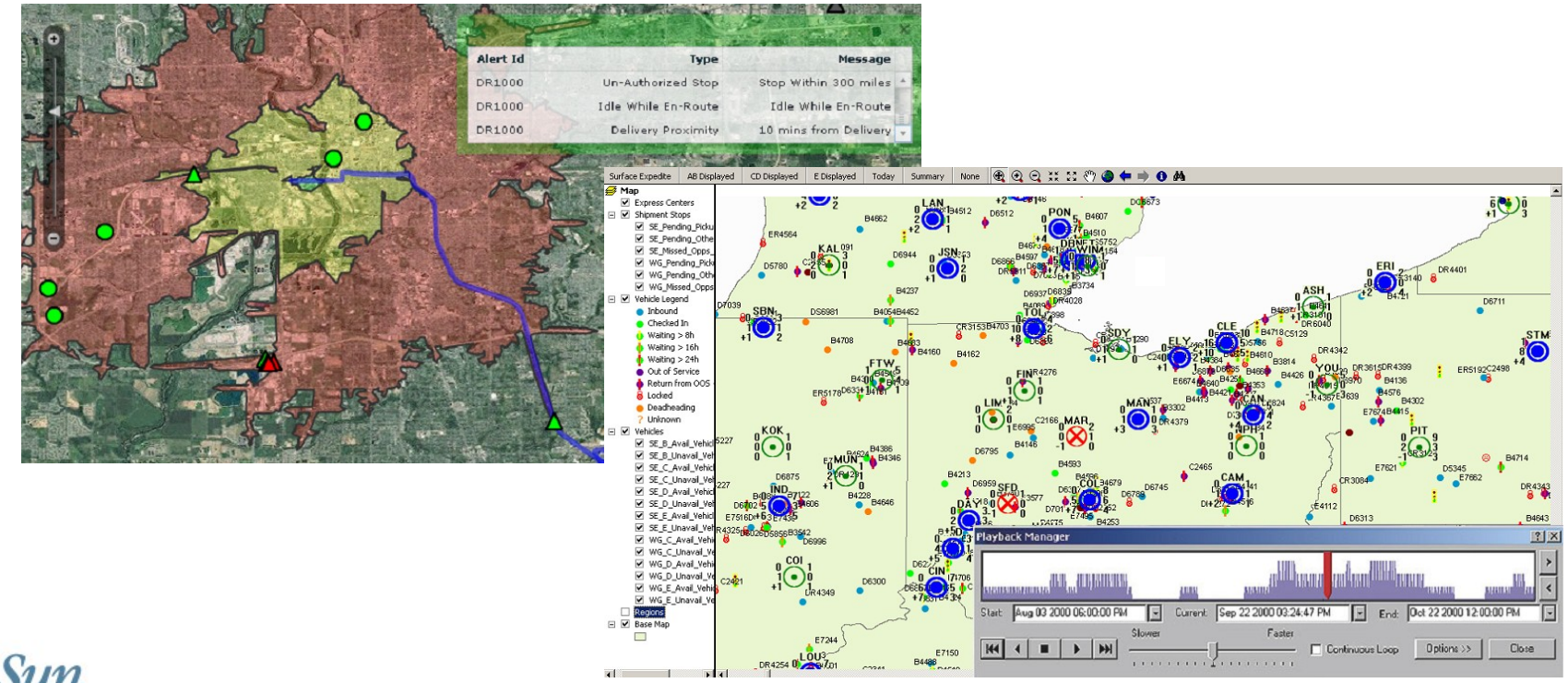
Characteristics of a CEP Engine
Event Processing

- Support processing high volume **Streams** of Events
  - Typically fed from **SOA endpoints** such as JMS Queues/Topics, Web Service calls, Databases, flat files, or sockets.
  - An **Event** is a record of state change. It is something that already happened, and the past cannot be changed, events are immutables.

CEP Applied

Stateful Rules Engine + CEP = Real-Time Intelligence

- **FedEx Custom Critical Case Studies**
  - En-Route Tracking Situational Awareness
  - Capacity Allocation Management




Drools
Business Logic
integration
Platform /
Guvnor
Expert
Fusion
Flow

Conditional Expressions
from

Using ‘from’ to reason over a nested list

```
rule “Find all the pets for a given owner"
when
    $owner : Person( name == “mark” )
             Pet( name == “rover” ) from $owner.pets
then
    …
end
```

Conditional Expressions
from

- **‘from’ can work on any expression, not just a nested field on a bound variable.**

Using ‘from’ with Hibernate Named Queries

```
rule “Find People for a given zip code"
when
    $zipCode : ZipCode()
    Person() from $hibernate.getNamedQuery( “Find People” )
                            .setParameters( [ “zipCode” : $zipCode ])
                            .list()
then
    …
end
```

Hibernate session

Conditional Expressions

accumulate

Accumulating Values

```
rule “accumulate"
when
    $acc : Number( intValue > 100 ) from accumulate
                                        ( Bus( color == “red”, $t : takings )
                                          sum( $t ) )
then
    print “sum is “ + $acc.sum;
end
```

Conditional Expressions

accumulate

- **Patterns and Conditional Expressions can be chained with ‘from’.**

Patterns and CE Chaining with ‘from’

```
rule “collect"
when
    $zipCode : ZipCode()
    $acc : Number( intValue > 100 ) from accumulate
                            ( Bus( color == “red”, $t : takings )
                              from $hibernate.getNamedQuery( “Find Buses” )
                                   .setParameters( [ “zipCode” : $zipCode ])
                                   .list(), sum( $t ) )
then
    print “sum is “ + $acc.sum;
end
```

Concurrent Event Stream Processing

Rules Engines do not scale for CEP

- **Rules Engines have a single point of insertion and are single threaded.**
- **CEP has concurrent Streams of Events.**

Rule Engine Scaling problem with CEP

```
ruleEngineSession.insert( event );    // Single point of entry
```

```
rule “Process Buy Order"
when
    $c : Customer( type == “VIP” )
    BuyOrderEvent( customer == $c )
then
    …
end
```

Patterns evaluate facts sequentially in a single thread.

Concurrent Event Stream Processing

entry-point = Scalable

- **Concurrent Event Stream Processing is made possible via named entry points.**

Using entry-point in a Rules-Engine for CEP

```
EntryPoint ep = session.getEntryPoint( “HomeBrokerEP” );
ep.insert( event );    // Now we can insert different streams concurrently
```

```
rule “Process Buy Order from Entry Point"
when
    $c : Customer( type == “VIP” )
    BuyOrderEvent( customer == $c ) from entry-point “HomeBrokerEP”
then
    …
end
```

When not specified uses the “default” entry-point

Patterns can now optionally specify their entry-point

Automatic Life-Cycle Management

- **Fact life-cycles must be managed by the user, so retractions are manual.**
- **Event life-cycles are automatically managed.**

Declaring a type as an Event

```
declare StockTick
    @role( event )
end;    // will be retracted when it is no longer needed
```

```
declare StockTick
    @role( event )
    @timestamp( timestampAttr )
    companySymbol : String
    stockPrice : double
    timestampAttr : long
end;
```

The declare statement can also specify an internal model that external objects/xml/csv map onto.
Drools supports Smooks and JAXB.

Temporal Operators

Reasoning over Time

- **Rule Engines do not have a rich enough set of temporal comparison operators**

Temporal Operator ‘after’ Detection

```
rule “Confirm Buy Acknowledge after Buy"
when
    $c : Customer( type == “VIP” )
    $oe : BuyOrderEvent( customer == $c )
         from entry-point “HomeBrokerEP”
    BuyAckEvent( relatedEvent == $oe.id, this after[1s, 10s] $oe )
         from entry-point “StockTrackerEP”
then
    …
end
```

BackAckEvent must occur between 1 and 10 seconds ‘after’ BuyOrderEvent

Temporal Operators
Reasoning over Time

Temporal Operator Not ‘after’ Detection

```
rule “Detect Non-Acknowledgement after Buy"
when
    $c : Customer( type == “VIP” )
    $oe : BuyOrderEvent( customer == $c )
                        from entry-point “HomeBrokerEP”
    not BuyAckEvent( relatedEvent == $oe.id, this after[1s, 10s] $oe )
                        from entry-point “StockTrackerEP”
then
    …
end
```

Existing Drools ‘not’ Conditional Elements
can be used to detect non-occurrence of events

Temporal Operators

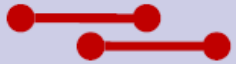
Reasoning over Time

- **Allow for detection, correlation, aggregation, and composition of Events.**
- **Temporal Constraint operators express relationships between point-in-time and interval-based events.**

Temporal Constraint Operators		
coincides before after meets metby	overlaps overlappedby during includes	starts startedby finishes finishedby

Temporal Operators

Reasoning over Time

	Point-Point	Point-Interval	Interval-Interval
A before B			
A meets B			
A overlaps B			
A finishes B			
A includes B			
A starts B			
A coincides B			

Sliding Time Windows and Aggregations

- **Rule Engines react to events happening now, there is no understanding of changes over time.**

```
StockTicker( symbol = “RHAT” )  over window:time ( 5s )
StockTicker( symbol = “RHAT” )  over window:length ( 10 )
```

- **Rules Engines do not deal with aggregations**

```
rule “Aggregate ticker price for RHAT over last 5 seconds"
when
    $n : Number( intValue > 100 )
                    from accumulate ( $s : StockTicker( symbol = “RHAT” )
                    over window:time( 5s ),
                    average( $s.price ) )
then
    …
end
```

Future Developments
Drools Chance

- **Reasoning with Imperfect Information**
  - Fuzzy Constraints
    - “Soft” constraints
    - Expressed in natural language
  - Probabilistic Constraints
    - Missing information
    - Forecast future events
  - Configurable Native Processing
    - Meta-information is processed by the Engine
  - Extended Language
    - New Evaluators and Operators

Future Developments
Drools Chance

- Example: Fuzzy constraints in Gradual rules

Gradual rule

```
rule "The later, the greater the fine"
when
      $due :ExpectedArrival ( $loc : location,
           $vid : vehicleId  ) from entry-point ”stopEP"
      $v : Vehicle ( vehicleId == $vid,
           ( position ~far $loc and @( type=”prod” ) speed ~slow )
           this ~after $due ) from entry-point ”vehicleEP"
           /* information known with certainty, but vague*/
then
   …   warn( “Possibility of delay :” + getConsequenceDegree() ); ...
end
```

Future Developments

Drools Chance

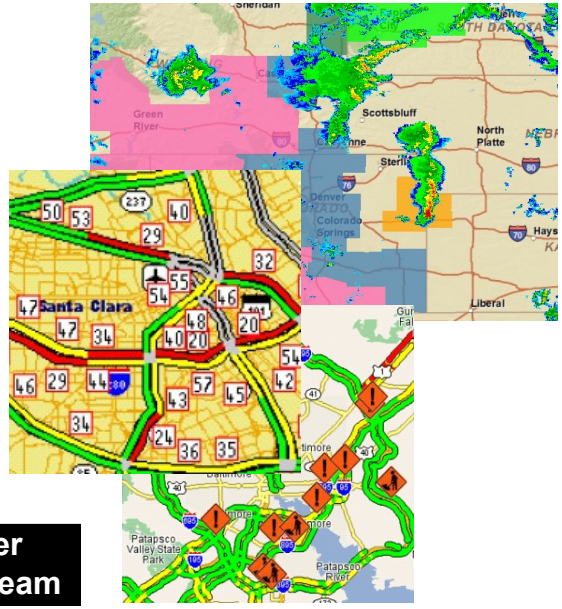
- Mixed constraints and logical entailment

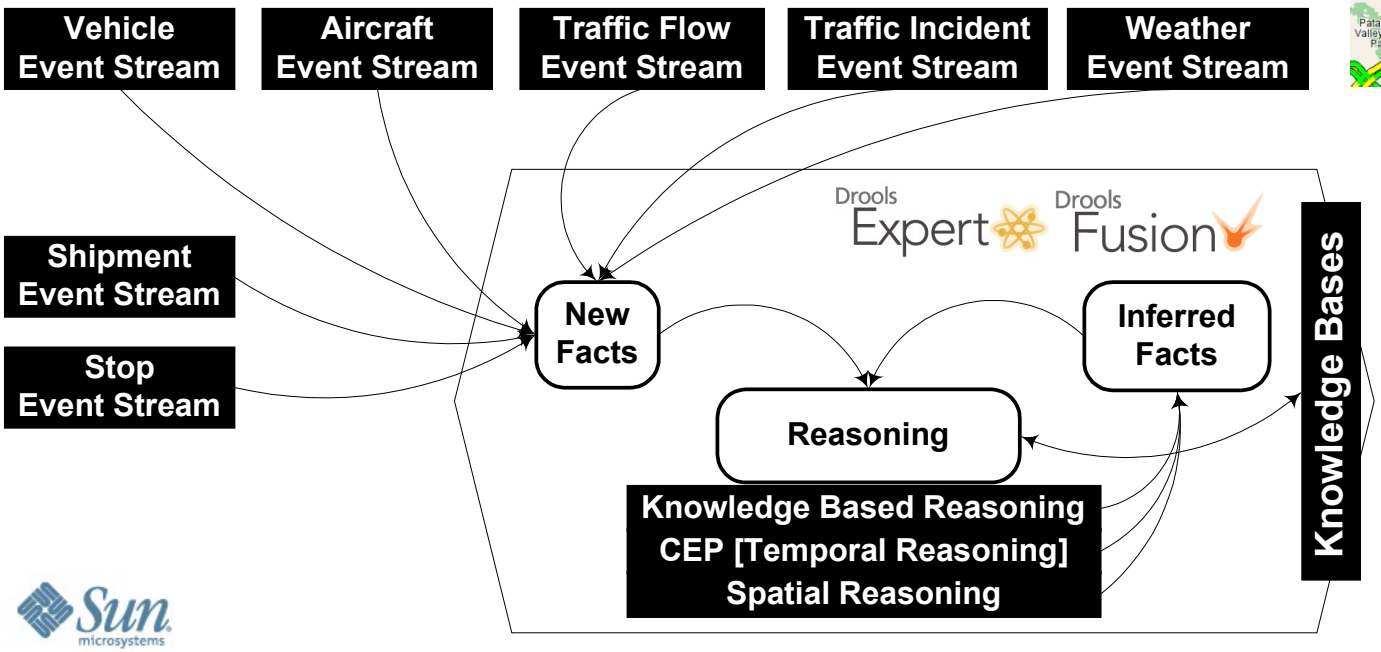
Mixed rules

```
rule "Incident"
when
      Incident( $loc : location)
            from entry-point ”IncidentEP"
      $r : Route( $loc memberOf locations )
then     inject( $r, ”hvy” );          end          //influences % of “~heavy”

rule "Route choice"
when
      $v : Vehicle ( $vid : vehicleId, $loc : location)
            from entry-point ”vehicleEP"
      $r : Route( start = $location,
            traffic ~heavy @( id=”hvy” ) )          //uncertain fuzzy evaluation
then     warn( “Traffic could be heavy :” )  ... end
```

Applying CEP at FedEx Custom Critical





Applying CEP at FedEx Custom Critical

Declaring point-in-time Vehicle Event

```
import com.fedex.enroutetracking.VehicleSensors;
declare VehicleSensors
    @role( event )
    @timestamp ( lastEventTime.time )
    @expires ( 1h30m ) // of the form:  [#d] [#h] [#m] [#s] [#[ms]]
end
```

Declaring interval-based TrafficIncident Event

```
import com.fedex.tracking.traffic.TrafficIncident;
declare TrafficIncident
    @role( event )
    @timestamp ( startTime )
    @duration ( incidentDuration )    // in milliseconds
end
```

Applying CEP at FedEx Custom Critical

Producing Events to an Entry-Point

```
KnowledgeBase kb = readKnowledgeBase();
StatefulKnowledgeSession ks = kb.newStatefulKnowledgeSession();
WorkingMemoryEntryPoint ep = ks.getWorkingMemoryEntryPoint(“vehsensEP”);
VehicleSensors sens = …;
vehicleEP.insert(sens);
```

Consuming Events from an Entry-Point

```
rule "Revise Stop ETAs based on new Vehicle Position"
when
    $vehicle : Vehicle ( $vid : vehicleId, $pos : position,
                         status == “EnRouteToStop” ) from entry-point ”vehsensEP"
    $stops : Stops ( vehicleId == $vid )
then
    $stops = EtaCalculator.reviseStopETAs( $vehicle, $stops );
    update( $stops );
end
```

Applying CEP at FedEx Custom Critical

- Detect and react to patterns in events for **Sliding Windows** of Interest (Time or Length).

Detecting Idle (Slow Moving) Vehicles

```
rule "Speed Idle En-Route Alert"
when
   $stats : SensorStatistics( $vid : vehicleId )
   $alerts : Alerts( vehicle == $vid )
   $vehicle : Vehicle ( vehicleId == $vid, inStopProximity == false )
   Number ( $avgSpeedMph : floatValue < 15 )
      from accumulate (
         VehicleSensors( vehicleId == $vid, inStopProximity == false,
                         $speedMph : speedMph )
         over window:time ( 15m ) from entry-point "vehsensEP",
         average ( $speedMph ) )
then
   $alerts.addAlert( new Alert (…) );
   update ( $alerts );
end
```

Accumulator functions:
sum, count, min, max
collect, [custom defined]

Applying CEP at FedEx Custom Critical
Deterministic Simulation Testing (Pseudo Clock)

- Reasoning over time requires a **Reference Clock**.
- **Session Clock** implements the GoF Strategy Pattern
- Rules Testing requires a **controlled** environment of input rules and facts.
- To Replay or Simulate scenarios it is necessary to control the **Flow of Time** via the **Pseudo Clock**.
- Regular Execution requires a **Real-Time clock**.

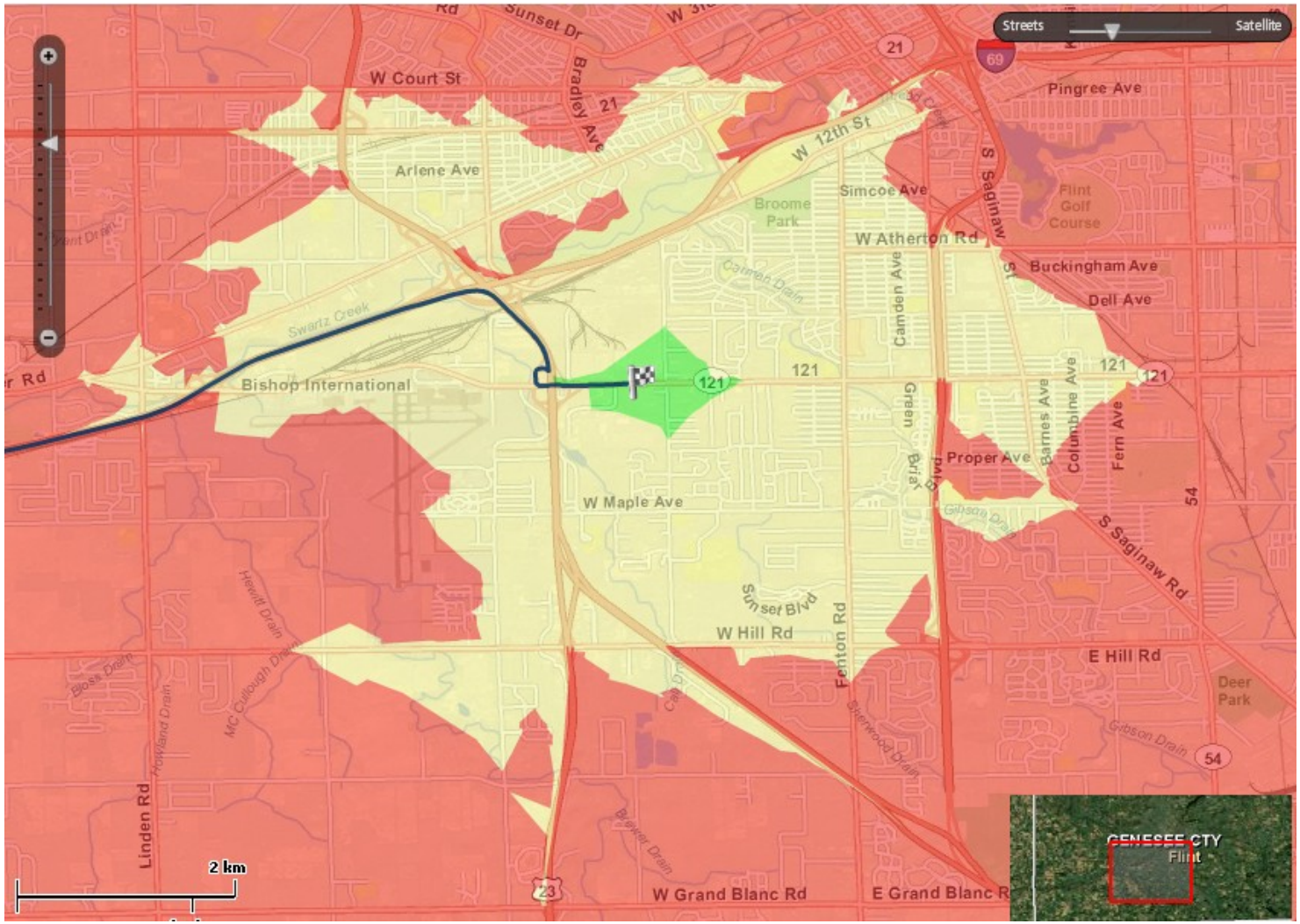
Applying CEP at FedEx Custom Critical
En-Route Tracking Case Study

En-Route Tracking Situational Awareness

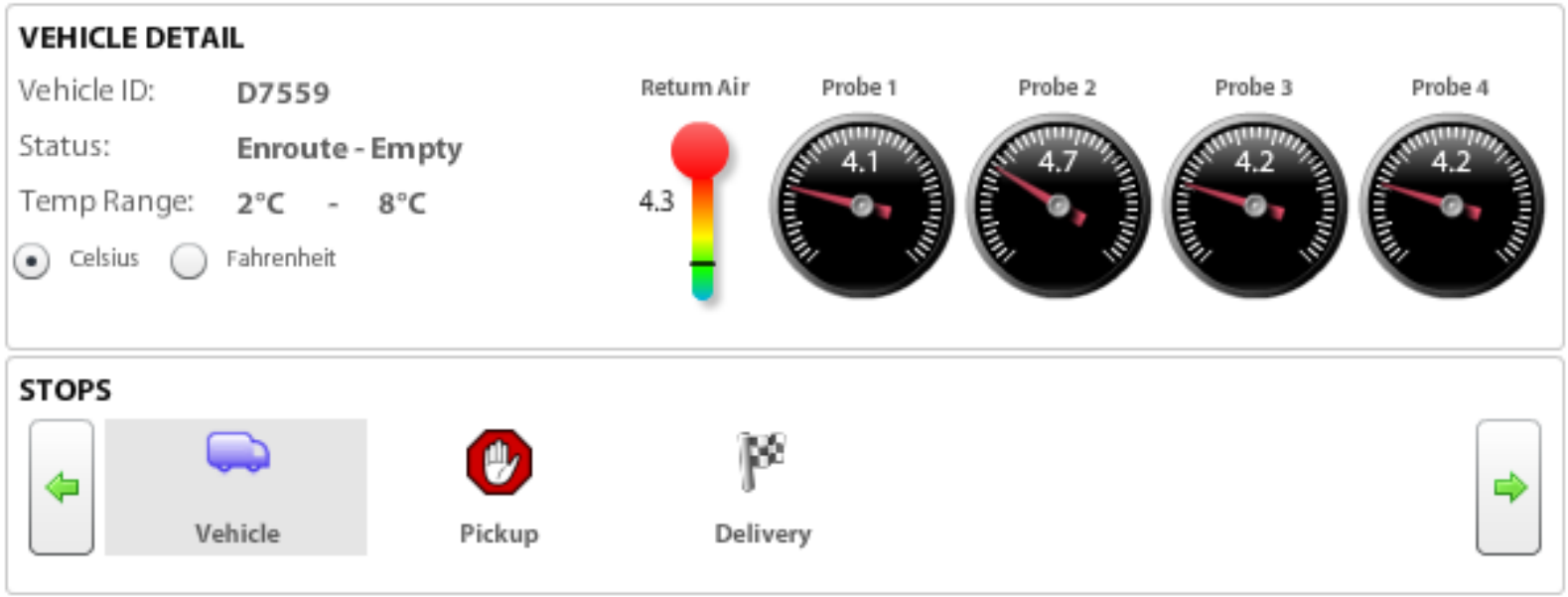
Demonstration,
Architecture Review,
and Code Walk-Through

Applying CEP at FedEx Custom Critical

Geo-Fencing based on Travel-Time Proximity

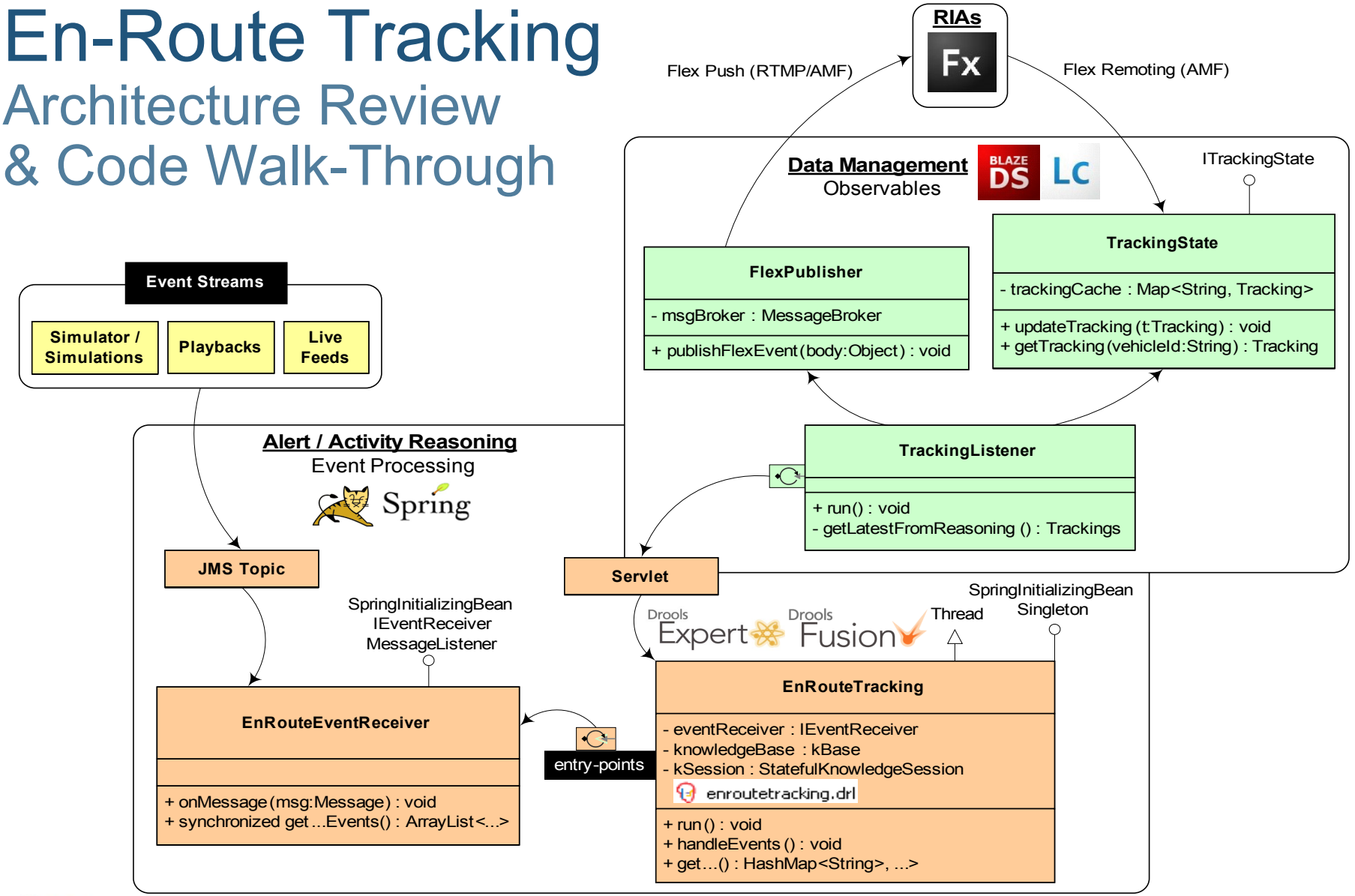


Applying CEP at FedEx Custom Critical
Temperature Tracking and Trend Detection



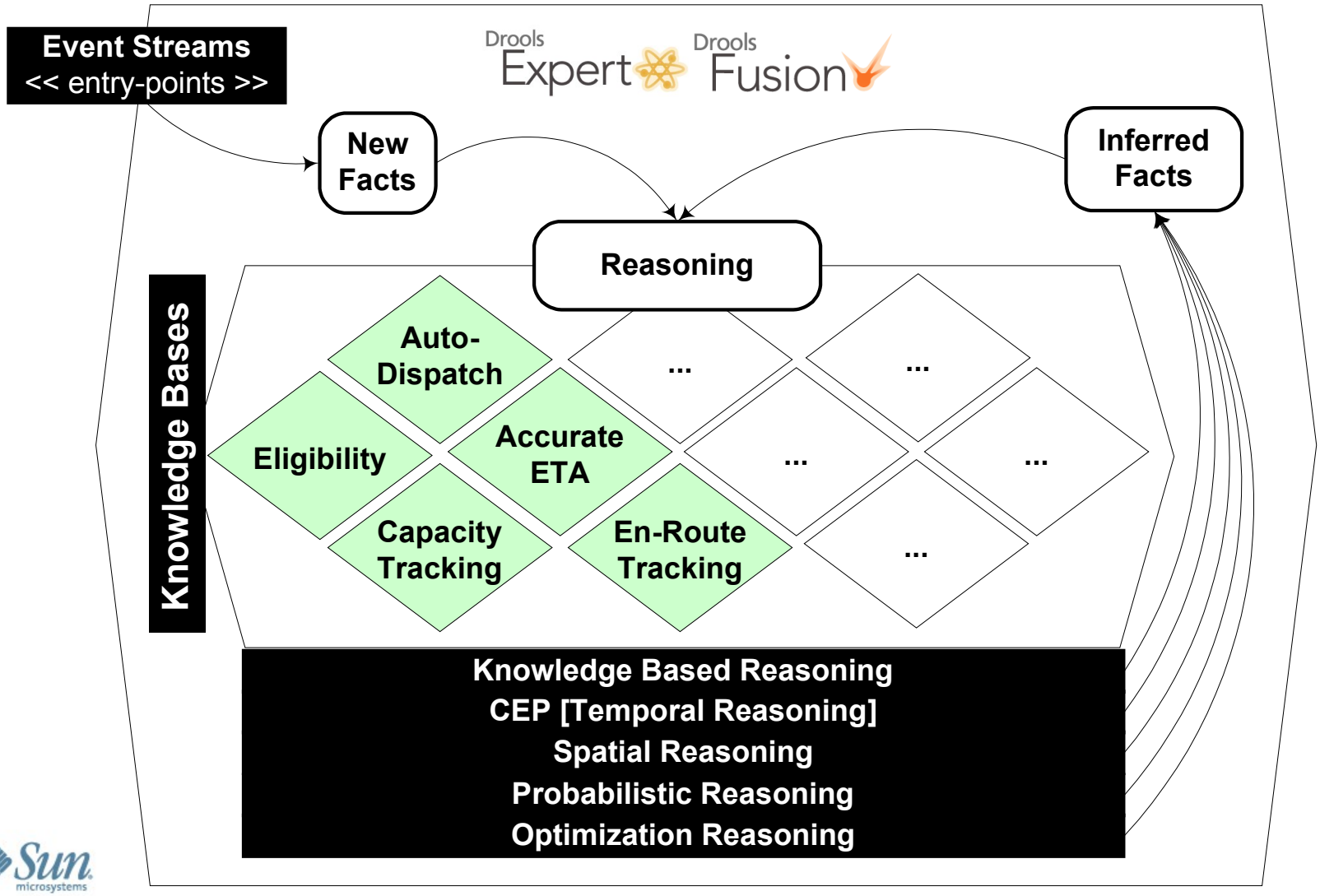
En-Route Tracking

Architecture Review & Code Walk-Through



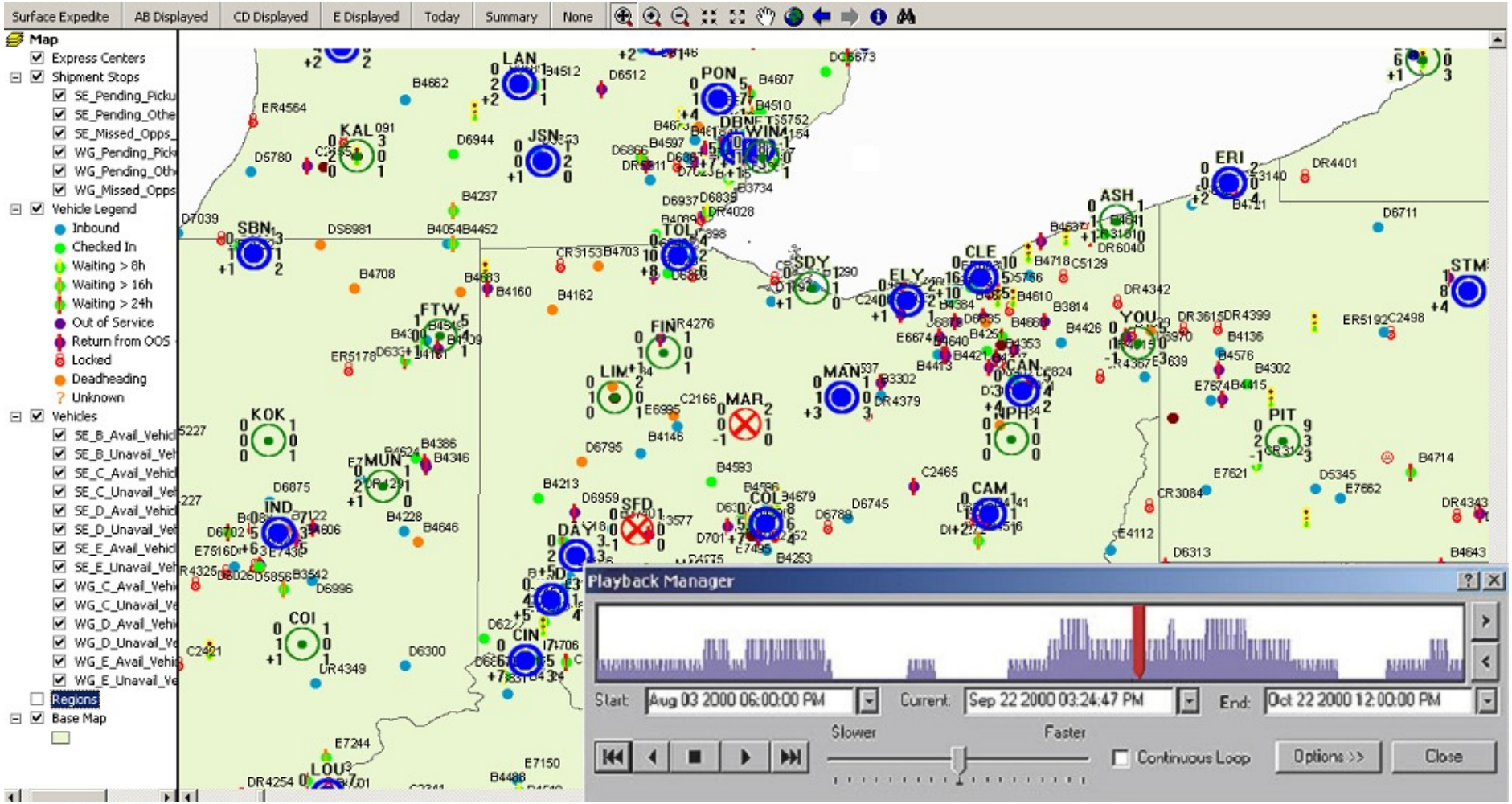
CEP + Stateful Rules Engine

Powerful when combined



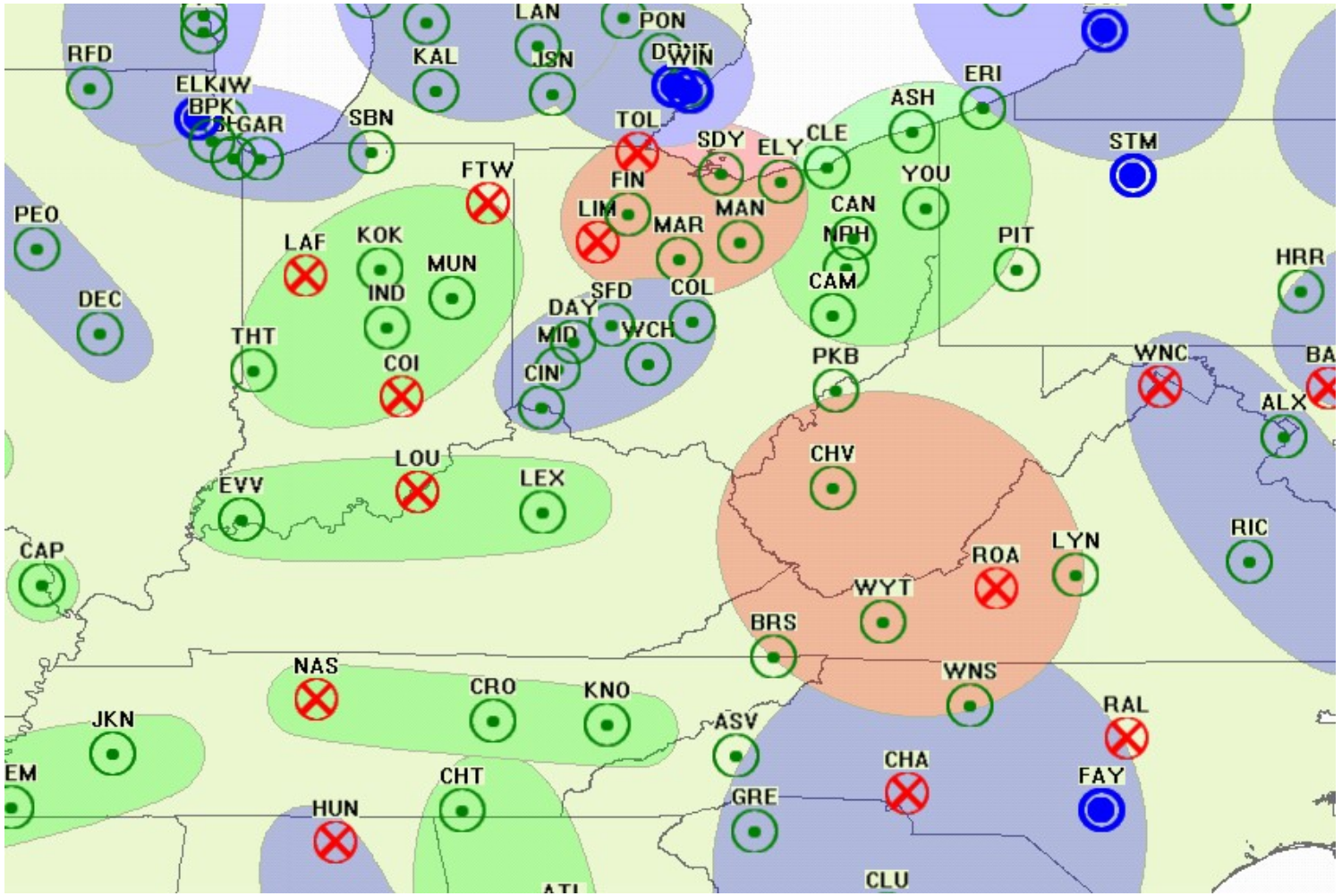
Applying CEP at FedEx Custom Critical

Capacity Allocation Management Case Study



Applying CEP at FedEx Custom Critical

Capacity Allocation Management Case Study

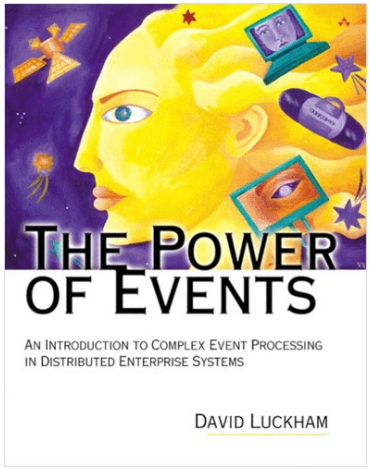


Spreading the Knowledge
Techniques to get the Answers

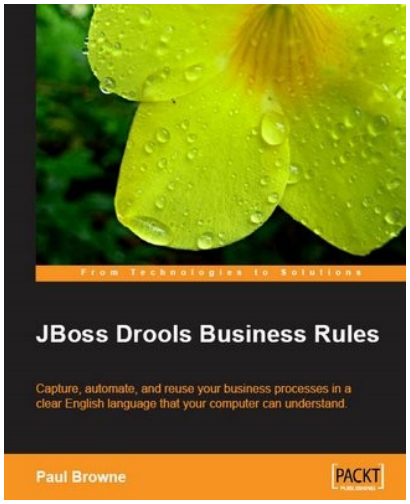
- Observe Observables
  - JMS Topics / Queues
  - RIA (Rich Internet Application) Push Protocols
- Ask Synchronously
  - Expose REST or SOAP Endpoints
  - Command Execution
  - Pipeline

Where to Learn More:

- Complex Event Processing
  - "The Power of Events" by David Luckham
    - See also http://complexevents.com/



  - Vendor Documentation:
    - Drools Fusion, Drools Expert
      - http://www.jboss.org/drools/documentation.html
      - http://www.jboss.com/drools
      - http://blog.athico.com
    - Oracle/BEA, IBM, Tibco
    - Sun, EsperTech, Coral8



JavaOne℠

Thank You

Adam Mollenkopf
adam.mollenkopf@fedex.com

Mark Proctor
mproctor@redhat.com

Sun microsystems