Stream and Complex Event Processing
Modeling Information Flow Processing Systems

G. Cugola      E. Della Valle      A. Margara

Politecnico di Milano  
cugola@elet.polimi.it
dellavalle@elet.polimi.it

Vrije Universiteit Amsterdam  
a.margara@vu.nl
Background

- Active Database Systems
- Data Stream Management Systems
- Event Based Systems & Complex Event Processing
Active DBMSs

• Standard DBMSs
  • Purely passive: Human-active database-passive (HADP)
  • Execution happens only when asked by clients (through queries)

• Active DBMSs
  • The reactive behavior moves (in part) from the application to the DB layer...
  • ...which executes Event Condition Action (ECA) rules
Active DBMSs

• As a DBMS extension
  • Rules may only refer to the internal state of the DB
• Closed DB applications
  • Rules may support the semantics of the application, but external sources of events are not allowed
• Open DB applications
  • Events may come from external sources
Data Stream Management Systems

- Data streams are (unbounded) sequences of time-varying data elements
- Represent:
  - an (almost) “continuous” flow of information
  - with the recent information being more relevant as it describes the current state of a dynamic system
Data Stream Management Systems

• The continuous nature of streams requires a paradigmatic change:
  • from persistent data stored and queried on demand
    • *One-time semantics*
  • to transient data consumed on the fly by continuous queries
    • *Continuous semantics*

• Continuous queries often operates through *windows*
Event-based systems

- Components collaborate by exchanging information about occurrent events. In particular:
  - Components *publish* notifications about the events they observe, or
  - they *subscribe* to the events they are interested to be notified about

- Communication is:
  - Purely message based
  - Asynchronous
  - Multicast
  - Implicit
  - Anonymous

Stream & Complex Event Processing - Models
The event dispatcher

- In event-based systems a special component of the architecture, the *event dispatcher*, is in charge of collecting subscriptions and routing event notifications based on such subscriptions.
  - For scalability reasons, its implementation can be distributed.
Complex Event Processing (CEP)

• CEP systems adds the ability to deploy *rules* that describe how composite events can be generated from primitive (or composite) ones

• Typical CEP rules search for *sequences of events*
  • Raise C if A→B

• Time is a key aspect in CEP
The current situation

• Back in 2007 CEP was already a hot topic...
• ... but having a good grasp of the area was rather hard
• As observed by Opher Etzion the area was looking like the “Tower of Babel”
  • Event Processing and the Babylon Tower – Event process thinking blog – Sept. 8, 2007
The current situation

- Several communities were contributing to the area...
- ... each bringing its own expertise and vocabulary...
- ... but often working in isolation
The current situation

• That was 2007. What about today?
• Things did not change much
  • From the “Event Process Thinking” blog
    [Which is] the relation between event processing and data stream management?
    1. They are aliases -- stream is just a collection of events, likewise, an event is just a member in a stream, and the functionality is the same
    2. Stream management is a subset of event processing -- there are different ways to do event processing, streams is one of them
    3. Event processing is a subset of stream management -- event streams is just one type of stream, but there are voice stream, video stream, ...
    4. Event processing and stream management are distinct and there is no overlapping between them
  • At the same time tool vendors are building tools that try to combine ¿ juxtapose ? different approaches
Our goal

- We would like to compare different systems in a precise way
- We would like to compare different approaches in a precise way
- We would like to help people coming from different areas communicate and compare their work with others
- We would like to isolate the open issues from those already solved
- We would like to precisely identify the challenges
- We would like to isolate the best part of the various approaches...
- ... finding a way to combine them

We need a *modeling framework* that could accommodate all the proposals

Forget your own vocabulary (temporarily)

- CEP, DSMSs, Stream reasoning, Active DBMSs...
  - All these terms hide a peculiar view of the domain we have in mind
  - With subtle, “unsaid” (and often unclear) differences

- Before learning something new we have to forget what we already know
The Information Flow Processing domain

- The **IFP engine** processes incoming *flows of information* according to a set of *processing rules*
  - Processing is “on line”
- **Sources** produce the incoming information flows, **sinks** consume the results of processing, **rule managers** add or remove rules
- Information flows are composed of *information items*
  - Items part of the same flow are neither necessarily ordered nor of the same kind
- Processing involve filtering, combining, and aggregating flows, item by item as they enter the engine
IFP: One name several incarnations

- A lot of applications
  - Environmental monitoring through sensor networks
  - Financial applications
  - Fraud detection tools
  - Network intrusion detection systems
  - RFID-based inventory management
  - Manufacturing control systems
  - ...

- Several technologies
  - Active databases
  - DSMSs
  - CEP Systems
One model, several models

- Different models to capture different viewpoints
  - Functional model
  - Processing model
  - Deployment model
  - Interaction model
  - Time model
  - Data model
  - Rule model
  - Language model
Functional model

- Implements the transport protocol to move information items along the net
- Acts as a demultiplexer

- Implements the transport protocol to move information items along the net
- Acts as a multiplexer

- Stream & Complex Event Processing - Models
A short digression

• We assume rules can be (logically) decomposed in two parts: C \(\rightarrow\) A
  • C is the condition
  • A is the action
• Example (in CQL):

\[
\text{Select IStream(Count(*)) From F1 [Range 1 Minute] Where F1.A > 0}
\]

• This way we can split processing in two phases:
  • The detection phase determines the items that trigger the rule
  • The production phase use those items to produce the output of the rule
Functional model

- Implements the detection phase
- Accumulates partial results into the history
- When a rule fires passes to the producer its action part

- Implements the production phase
- Uses the items in Seq as stated in action A

- Some systems allows rules to combine flowing items with items previously stored into a (read only) storage

- If present models the ability of performing *recursive processing* building hierarchies of items

- Optional component
- Periodically creates special information items holding current time
- Its presence models the ability of performing periodic processing of inputs

- Some systems allows rules to be added or removed at processing time

- Some systems allows rules to combine flowing items with items previously stored into a (read only) storage

- If present models the ability of performing *recursive processing* building hierarchies of items

- Optional component
- Periodically creates special information items holding current time
- Its presence models the ability of performing periodic processing of inputs

- Some systems allows rules to be added or removed at processing time

- Some systems allows rules to combine flowing items with items previously stored into a (read only) storage

- If present models the ability of performing *recursive processing* building hierarchies of items

- Optional component
- Periodically creates special information items holding current time
- Its presence models the ability of performing periodic processing of inputs
Functional model: Considerations

- The detection-production cycle
  - Fired by a new item I entering the engine through the Receiver
    - Including those periodically produced by the Clock, if present
  - Detection phase: Evaluates all the rules to find those enabled
    - Using item I plus the data into the Knowledge base, if present
    - The item I can be accumulated into the History for partially enabled rules
    - The action part of the enabled rules together with the triggering items (A+Seq) is passed to the producer
  - Production phase: Produces the output items
    - Combining the items that triggered the rule with data present in the Knowledge base, if present
    - New items are sent to subscribed sinks (through the Forwarder)...
    - ...but they could also be sent internally to be processed again (recursive processing)
    - In some systems the action part of fired rules may also change the set of deployed rules
Functional model: Considerations

- Maximum length of Seq a key aspect
  - $1 \approx$ PubSub
  - Bounded $\Rightarrow$
    - CQL like languages without time based windows
    - Pattern based languages without a Kleene+ operator
- Other key aspects that impact expressiveness
  - Presence of the Clock
    - Models the ability to process rules periodically
    - Available in almost half of the systems reviewed
    - Most Active DBMSs and DSMSs but few CEP systems
Functional model: Considerations

- Presence of the Knowledge base
  - Only available in systems coming from the database community
- Presence of the looping flow exiting the Producer
  - Models the ability of performing recursive processing
  - Half CEP systems have it
  - All Active DBMSs but very few DSMSs have it
    - They have nested rules
- Support to dynamic rule change
  - Few systems support it
  - Can be implemented externally...
    - Through sinks acting also as rule managers
  - ...but we think it is nice to have it internally
The semantics of processing

- What determines the output of each detection-production cycle?
  - The new item entering the engine
  - The set of deployed rules
  - The items stored into the History
  - The content of the Knowledge Base
- Is this enough?
- Example (in Padres and CQL):
  - `Smoke && Temp>50`
  - `Select IStream(Smoke.area)
    From Smoke[Rows 30 Slide 10], Temp[Rows 50 Slide 5]
    Where Smoke.area = Temp.area AND Temp.value > 50`
Processing model

• Three policies affect the behavior of the system
  • The selection policy
  • The consumption policy
  • The load shedding policy
Selection policy

- Determines if a rule fires once or multiple times and the items actually selected from the History

Example:
Selection policy: Considerations

• Most systems adopt a multiple selection policy
  • It is simpler to implement
  • Is it adequate?
    • Example: Alert fire when smoke and high temperature in a short time frame
      – If 10 sensors read high temperature and immediately afterward one detects smoke I would like to receive a single alert, not 10

• A few systems allow this policy to be programmed...
• ...some of them on a per-rule base
  • E.g., Amit, T-Rex
Selection policy: The TESLA case

• TESLA (Trio-based Event Specification Language): the T-Rex language
  • A rule language for CEP. Tries to combine expressiveness and efficiency
  • Has a formally defined semantics
    • Expressed in Trio, a Metric Temporal Logic (see DEBS 2010)
• Allows rule managers to choose their own selection policy on a per rule base
  • Example: Multiple selection
    ```plaintext
    define Fire(area: string, measuredTemp: double) 
    from Smoke(area) and 
    each Temp(area) within 1 min. from Smoke 
    where area=Smoke.area and measuredTemp=Temp.value
    ```
  • Example: Single selection
    ```plaintext
    define Fire(area: string, measuredTemp: double) 
    from Smoke(area) and 
    last Temp(area and val>50) within 1min. from Smoke 
    where area=Smoke.area and measuredTemp=Temp.val
    ```
  • Alternatively you may use:
    • first...within
    • n-first...within  n-last...within
Consumption policy

• Determines how the history changes after firing of a rule ⇒ what happens when new items enter the Decider

• Example:

![Diagram of Consumption policy]

Stream & Complex Event Processing - Models 29
Consumption policy: Considerations

- Most systems couple a multiple selection policy with a zero consumption policy
  - This is the common case with DSMSs, which use (sliding) windows to select relevant events
    - Example (in CQL)
      ```
      Select IStream(Smoke.area)
      From Smoke[Range 1 min], Temp[Range 1 min]
      Where Smoke.area = Temp.area AND Temp.val > 50
      ```
  - The systems that allow the selection policy to be programmed often allow the consumption policy to be programmed, too
    - E.g., Amit, T-Rex
Consumption policy: The TESLA case

• Zero consumption policy
  
  • define Fire(area: string, measuredTemp: double)
  from Smoke(area=$a) and
  each Temp(area=$a and val>50)
  within 1min. from Smoke
  where area=Smoke.area and measuredTemp=Temp.value

• Selected consumption policy
  
  • define Fire(area: string, measuredTemp: double)
  from Smoke(area=$a) and
  each Temp(area=$a and val>50)
  within 1min. from Smoke
  where area=Smoke.area and measuredTemp=Temp.value consuming Temp
Load shedding policy

• Problem: How to manage bursts of input data
• It may seem a system issue
  • i.e., an issue to solve into the Receiver
• But it strongly impacts the results produced
  • i.e., the “semantics” of the rules
• Accordingly, some systems allows this issue to be determined on a per-rule basis
  • e.g., Aurora allows rules to specify the expected QoS and sheds input to stay within limits with the available resources
  • Conceptually the issue is addressed into the decider
Deployment model

• IFP applications may include a large number of sources and sinks
  • Possibly dispersed over a wide geographical area
• It becomes important to consider the deployment architecture of the engine
  • How the components of the functional model can be distributed to achieve scalability
Deployment model

- Centralized
- Clustered
- Networked
- Distributed

Sources → IFP Engine → Sinks

Information Flows

Rules

Rule managers

Stream & Complex Event Processing - Models
Deployment Model

• Most existing systems adopt a centralized solution
• When distributed processing is allowed, it is usually based on clustered solutions
• A few systems have recognized the importance of networked deployment for some applications
  • E.g. Microsoft StreamInsight (part of SQLServer)
    • Filtering near sources
    • Aggregation and correlation in-network
    • Analytics and historical data in a centralized server/cluster
• In most cases, deployment/configuration is not automatic
Deployment model

- Automatic distribution of processing introduces the _operator placement_ problem
- Given a set of rules (composed of operators) and a set of nodes
  - How to split the processing load
  - How to assign operators to available nodes
- In other words (Event Processing in Action)
  - Given an _event processing network_
  - How to map it onto the physical network of nodes
Operator placement

- The operator placement problem is still open
  - Several proposals
    - Often adopting techniques coming from the Operational Research
  - Difficult to compare solutions and results
    - Even in its simplest form the problem is NP-hard

[more on this later during the course]
More on deployment model

• Operator placement is only part of the problem
• Other issues
  • How to build the network of nodes?
  • How to maintain it?
  • How to gather the information required to solve the operator placement problem?
  • How to actually “place” the operators?
  • How to “replace” them when the situation changes?
    • New rules added, old rules removed...
    • …new sources/sinks
Deployment model and dynamics

• How to cope with mobile nodes?
  • Mobile sinks and sources...
  • ...but also mobile “processors”
• The issue is relevant
  • We leave in a mobile world
• Very few proposals
• A lot of work in the area of pure publish/subscribe
  • Several works published in DEBS, not to mention other major conferences/journals
• May we reuse some of this work?
Interaction Model

- It is interesting to study the characteristics of the interactions among the main component of an IFP system
  - Who starts the communication?
Interaction Model

Sources

IFP Engine

Sinks

Observation Model
- Push
- Pull

Forwarding Model
- Push
- Pull

Notification Model
- Push
- Pull

Stream & Complex Event Processing - Models
Time Model

• Relationship between information items and passing of time
• Ability of an IFP system to associate some kind of happened-before (ordering) relationship to information items
• We identified 4 classes:
  1. Stream-only
  2. Causal
  3. Absolute
  4. Interval
Stream-Only Time Model

- Used in original DSMSs
- Timestamps may be present or not
- When present, they are used only to order items before entering the engine, then they are forgotten
- *They are not exposed to the language*
  - With the exception of windowing constructs
- Ordering in output streams is conceptually separate from the ordering in input streams

**CQL/Stream**

Select DStream(*)

From F1[Rows 5], F2[Range 1 Minute]

Where F1.A = F2.A
Causal Time Model

• Each item has a label reflecting some kind of causal relationship

• Partial order

• E.g. Rapide
  • An event is causally ordered after all events that led to its occurrence

Gigascope
Select count(*)
From A, B
Where A.a-1 <= B.b and
    A.a+1 > B.b
A.a, B.b monotonically increase
Absolute Time Model

- Information items have an associated timestamp
- Defining a single point in time w.r.t. a (logically)unique clock
  - Total order
- Timestamps are fully exposed to the language
- Information items can be timestamped at source or entering the engine

**TESLA/T-Rex**
Define Fire(area: string, measuredTemp: double)
From Smoke(area=$a) and last Temp(area=$a and value>45) within 5 min. from Smoke
Where area=Smoke.area and measuredTemp=Temp.value
Interval Time Model

- Used for events to include “duration”
  - SnoopIB, Cayuga, NextCEP, …
- At a first sight, it is a simple extension of the absolute time model
  - Timestamps with two values: start time and end time
- However, it opens many issues
  - What is the successor of an event?
  - What is the timestamp associated to a composite event?
Interval Time Model

• Which is the immediate successor of A?
  • Choose according to end time only: B
    • But it started before A!
  • Exclude B: C, D
    • Both of them?
    • Which of them?
• No other event strictly between A and its successor: C, D, E
  • Seems a natural definition
  • Unfortunately we loose associativity!
    – \( X \rightarrow (Y \rightarrow Z) \neq (X \rightarrow Y) \rightarrow Z \)
  • May impede some rule rewriting for processing optimizations

Stream & Complex Event Processing - Models
Interval Time Model

• “What is “Next” in event processing?” by White et. Al
  • Proposes a number of desired properties to be satisfied by the “Next” function
  • There is one model that satisfies them all
    • Complete History
  • It is not sufficient to encode timestamps using a couple of values
  • Timestamps of composite events must embed the timestamps of all the events that led to their occurrence
  • Possibly, timestamps of unbounded size
    • In case of unbounded Seq
Data Model

- Studies how the different systems
  - Represent single data items
  - Organize them into data flows

### Data Items
- **Nature of Items**
  - Generic Data
  - Event Notifications

- **Format**
  - Records
  - Tuples
  - Objects
  - ...

- **Support for Uncertainty**

### Data Flows
- **Homogeneous**
- **Heterogeneous**
Nature of Items

- The meaning we associate to information items
  - Generic data
  - Event notifications
- Deeply influences several other aspects of an IFP system
  - Time model !!!
  - Rule language
  - Semantics of processing
- Heritage of the heterogeneous backgrounds of different communities

Data

Data Items

Nature of Items
- Generic Data
- Event Notifications

Format
- Records
- Tuples
- Objects
- ...

Support for Uncertainty

Data Flows
- Homogeneous
- Heterogeneous
Nature of Items

CQL/Stream (Generic Data)

Select IStream(*)
From F1[Rows 5],
F2[Range 1 Minute]
Where F1.A = F2.A

TESLA/T-Rex (Event Notifications)

Define Fire (area: string,
measuredTemp: double)
From Smoke(area=$a)and last
Temp(area=$a and value>45)
within 5 min.
from Smoke
Where area=Smoke.area and
measuredTemp=Temp.value
Format of Items

- How information is represented
- Influences the way items are processed
  - E.g., Relational model requires tuples

Data

Data Items

Nature of Items
- Generic Data
- Event Notifications

Format
- Records
- Tuples
- Objects
- ...

Support for Uncertainty

Data Flows
- Homogeneous
- Heterogeneous
Support for Uncertainty

- Ability to associate a degree of uncertainty to information items
  - To the content of items
    - Imprecise temperature reading
  - To the presence of an item (occurrence of an event)
    - Spurious RFID reading
- When present, probabilistic information is usually exploited in rules during processing

[more on this later]
Data Flows

• Homogeneous
  • Each flow contains data with the same format and “kind”
    • E.g. Tuples with identical structure
  • Often associated with “database-like” rule languages

• Heterogeneous
  • Information flows are seen as channels connecting sources, processors, and sinks
  • Each channel may transport items with different kind and format
Rule Model

- Rules are much more complex entities than data items
- Large number of different approaches
  - Already observed in the previous slides
- Looking back to our functional model, we classify them into two macro classes
  - Transforming rules
  - Detecting rules
Transforming Rules

- Do not present an explicit distinction between detection and production
- Define an execution plan combining *primitive operators*
- Each operator transforms one or more input flows into one or more output flows
- The execution plan can be defined
  - explicitly (e.g., through graphical notation)
  - implicitly (using a high level language)
- Often used with homogeneous information flows
  - To take advantage of the predefined structure of input and output

Type of Rules
- Transforming Rules
- Detecting Rules

Support for Uncertainty

Detecting Rules

- Present an explicit distinction between detection and production
- Usually, the detection is based on a logical predicate that captures *patterns* of interest in the history of received items
Support for Uncertainty

- Two orthogonal aspects
  - Support for uncertain input
    - Allows rules to deal with/reason about uncertain input data
  - Support for uncertain output
    - Allows rules to associate a degree of uncertainty to the output produced

[more on this later]
Language Model

- Following the rule model, we define two classes of languages:
  - **Transforming languages**
    - Declarative languages
    - Imperative languages
  - Detecting languages
    - Pattern-based

- Specify operations to
  - Filter
  - Join
  - Aggregate
  - input flows ...
  - ... to produce one or more output flows
Language Model

• Following the rule model, we define two classes of languages:
  • Transforming languages
    • Declarative languages
    • Imperative languages
  • Detecting languages
    • Pattern-based
• Specify the expected result rather than the desired execution flow
• Usually derive from relational languages
  • Relational algebra / SQL

**CQL/Stream:**
Select IStream(*)
From F1[Rows 5], F2[Rows 10]
Where F1.A = F2.A
Language Model

• Following the rule model, we define two classes of languages:
  • Transforming languages
    • Declarative languages
    • Imperative languages
  • Detecting languages
    • Pattern-based

• Specify the desired execution flow
• Starting from primitive operators
  • Can be user-defined
• Usually adopt a graphical notation
Imperative Languages

Aurora (Boxes & Arrows Model)
Hybrid Languages

Oracle CEP
Language Model

• Following the rule model, we define two classes of languages:
  • Transforming languages
    • Declarative languages
    • Imperative languages
  • Detecting languages
    • Pattern-based

• Specify a firing condition as a pattern
• Select a portion of incoming flows through
  • Logic operators
  • Content / timing constraints
• The action uses selected items to produce new knowledge
Detecting Languages

**TESLA / T-Rex**

Define Fire(area: string, measuredTemp: double)

From Smoke(area=$a) and last Temp(area=$a and value>45) within 5 min. from Smoke

Where area=Smoke.area and measuredTemp=Temp.value
Language Model

• Different syntaxes / constructs / operators
• Comparison of languages semantics and expressiveness still an open issue
• Our approach:
  • Review all operators encountered in the analysis of systems
  • Specifying the classes of languages adopting them
  • Trying to capture some semantics relationship
    • Among operators
Language Model

- Single-Item operators
  - Selection operators
    - Filter items according to their content
  - Elaboration operators
    - Projection
      - Extracts a part of the content of an item
    - Renaming
      - Changes the name of a field in languages based on records or tuples
- Present in all languages
- Defined as primitive operators in imperative languages
- Declarative languages inherit selection, projection, and renaming from relational algebra

```sql
Select RStream (I.Price as HighPrice)
From Items[Rows 1] as I
Where I.Price > 100
```

- Selection
- Projection
- Renaming
Language Model

• Single-Item operators
  • Selection operators
    • Filter items according to their content
  • Elaboration operators
    • Projection
      – Extracts a part of the content of an item
    • Renaming
      – Changes the name of a field in languages based on records or tuples

• Pattern-based languages
  • Selection inside the condition part (pattern)
  • Elaboration as part of the action

Define ExpensiveItem
  (highPrice: double)
From Item(price>100)
Where highPrice = price
Language Model

- Logic Operators
  - Conjunction
  - Disjunction
  - Repetition
  - Negation

- Explicitly present in pattern-based languages

PADRES

\[(A \& B) \mid \mid (C \& D)\]
Language Model

- Logic Operators
  - Conjunction
  - Disjunction
  - Repetition
  - Negation

- Some logic operators are blocking
  - Express pattern whose validity cannot be decided into a bounded amount of time
    - E.g., Negation
  - Used in conjunction with windows

Define Fire()
From Smoke(area=$a) and not Rain(area=$a) within 10 min from Smoke
Language Model

- Logic Operators
  - Conjunction
  - Disjunction
  - Repetition
  - Negation

- Traditionally, logic operators were not explicitly offered by declarative and imperative languages
- However, they could be expressed as transformation of input flows

```sql
Select IStream (F1.A, F2.B)
From F1 [Rows 10],
F2 [Rows 20]
```
Language Model

• Sequences
  • Similar to logic operators
  • Based on timing relations among items

• Present in almost all pattern-based languages

Define Fire()
From Smoke(area=$a) and last Temp(area=$a and value>45) within 5 min. from Smoke
Language Model

• Sequences
  • Similar to logic operators
  • Based on timing relations among items

• Traditionally, transforming languages did not provide sequences explicitly
  • Could be expressed with an explicit reference to timestamps
    • If present inside items

Select IStream (F1.A, F2.B)
From F1 [Rows 10], F2 [Rows 20]
Where F1.timestamp < F2.timestamp

Impose timestamp order
Language Model

• Iterations
  • Express possibly unbounded sequences of items ...
  • ... satisfying an *iterating* condition
    • Implicitly defines an ordering among items

SASE+

PATTERN SEQ(Alert a, Shipment+ b[ ])
WHERE skip_till_any_match(a, b[ ])
    { a.type = 'contaminated' and
      b[1].from = a.site and b[i].from = b[i-1].to
    } WITHIN 3 hours
Language Model

• Logic operators, sequences, and iterations *traditionally* not offered by transforming languages

• And now?
  - Current trend:
    - Embed patterns inside declarative languages
    - Especially adopted in commercial systems

```
Esper
Select A.price
From pattern
    [every (A \rightarrow (B or C))]
Where A.price > 100
```

• IMO: difficult to write / understand
• Mailing list of Esper!
Language Model

• Windows
  • Kind:
    • Logical (Time-Based)
    • Physical (Count-Based)
    • User-Defined

Logical
Select IStream(Count(*))
From F1[Range 1 Minute]

Physical
Select IStream(Count(*))
From F1[Rows 50 Slide 10]
Language Model

- Windows are used to limit the scope of blocking operators
- They are generally available in declarative and imperative languages
- They are not present in all pattern-based languages
  - Some of them do not include blocking operators
  - Some of them “embed” windows inside operators
    - Making them unblocking

CEDR
EVENT Test-Rule
WHEN UNLESS(A, B, 12 hours)
WHERE A.a < B.b
Language Model

• **Windows movement**
  • *Fixed*: do not move at all
  • *Landmark*: have a fixed lower bound, while the upper bound advances every time a new information item enters the system
    • E.g., all items since 1/1/2013
  • *Sliding*: have a fixed size, both lower and upper bounds advance when new items enter the system
  • *Pane*: both the lower and the upper bounds move by k elements, as k elements enter the system
    • K is smaller than the window size
  • *Tumble*: same as above
    • K is greater or equal to the window size
Language Model

• Flow management operators
  • Required by declarative and imperative languages to merge, split, organize, and process incoming flows of information

Flow Management Operators

- Join
- Order By
- Group By
- Flow Creation
- Bag Operators
- Duplicate

- Union
- Except
- Intersect
- Remove-duplicates
Language Model

• Parameterization
  • Allows the binding of different information items based on their content
  • Offered implicitly by declarative and imperative languages
    • Through a combination of join and selection
  • Offered as an explicit operator in pattern-based languages

CQL / Stream
Select IStream (F1.A, F2.B)
From F1 [Rows 10], F2 [Rows 20]
Where F1.A > F2.B

TESLA / T-Rex
Define Fire()
From Smoke(area=$a) and last Temp(area=$a and value>45)
within 5 min. from Smoke
Language Model

Aggregates

Scope

• Detection Aggregates
• Production Aggregates

Definition

• Predefined
• User-defined

Define Fire(area: string, measuredTemp: double)
From Smoke(area=$a) and 45 < Avg(Temp(area=$a).value within 5 min. from Smoke)
Where area=Smoke.area and measuredTemp=Temp.value

Define Fire(area: string, measuredTemp: double)
From Smoke(area=$a) and last Temp(area=$a and value>45) within 5 min. from Smoke)
Where area=Smoke.area and measuredTemp=Avg(Temp(area=$a).value) within 1 hour from Smoke
Discussion
Abstraction

- IFP languages and systems offer a level of abstraction over flowing information
  - Similar to the role SQL / DBMSs play for static data
- The heterogeneity of solutions suggests that finding the “right” abstraction is still an open issue
  - Several open questions
Abstraction

• Which are the requirements of applications?
  • According to the results of EPTS survey (2010)
    • Simple operations (mainly aggregate / join)
    • Few sources (mainly 1-10, and 10-100)
    • Low rates (mainly 10-100, and 100-1000 event/s)
    • Data coming from DBMSs / files
Abstraction

• How can we explain these answers?
  • IFP systems used as “enhanced” DBMSs
    • IMO, companies are using the technology they know
      – E.g. Language: no need for patterns or no knowledge about patterns?
    • Features vs Simplicity/Integration
  • This is the “consolidated” part of IFP systems
    • IMO, raising the level of abstraction would make IFP systems more appealing for a larger audience
      – Is it possible to combine advanced features and ease of use?
Abstraction

- Other details are present in the EPTS survey
  - Most of the people was referring to technologies that were “already deployed” or “in development”
  - The systems were chosen based on the trust on the vendor
- This suggests that the last (research) advancements in event processing may be still unknown
  - They lack a solid implementation ...
  - ... that can easily work side by side with existing data processing systems
Abstraction

• How many kinds of languages / abstractions?
  • Traditionally, two
    • Transforming rules
      – Based on the Stream model
    • Detecting rules
      – Based on patterns
  • Often merged together in commercial systems
    • “Merged together” does not mean “organically combined”
    • The underlying model is usually derived from the Stream model
      – Good integration with relational languages
      – Difficult to integrate with patterns
      – Difficult to understand the semantics of rules

• Can we offer a better model / formalism?
Abstraction

• Do we really need “One abstraction to rule ‘em all”? 
  • Again, we need to better understand application requirements ...
  • ... but we also need to better analyze the expressiveness and semantics of languages!

• In our survey, we offer a description of existing operators

• Open issues:
  • How do operators contribute to the overall expressiveness of languages?
  • Is it possible to find a “minimal set” of operators to organically combine the capabilities of transforming and detecting rules?
Our Proposal: TESLA

• “One size fits all” language does not exist
  • At least, we have to find it yet
• We started from the following requirements
  • Focus specifically on events
    • Not generic data processing
  • Define a clear and precise semantics
    • Issues of selection and consumption policies
    • General issue of formal semantics
  • Be expressive
    • Useful for applications
• Keep it simple
  • Easy to read and write
Define \( CE(Att_1 : \text{Type}_1, \ldots, Att_n : \text{Type}_n) \)
From Pattern
Where \( Att_1 = f_1(\ldots), \ldots, Att_n = f_n(\ldots) \)
Consuming \( e_1, \ldots, e_m \)
Patterns in TESLA

- Selection of a single event
  - $A(x>10)$
  - Timer()
- Selection of sequences
  - $A(x>10)$ and each $B$ within 5 min from $A$
  - $A(x>10)$ and last $B$ within 5 min from $A$
  - $A(x>10)$ and first $B$ within 5 min from $A$
  - Generalization
    - n-first / n-last
Patterns in TESLA

• TESLA allows *-within operators to be composed with each other:
  • In chains of events
    • A and each B within 3 min from A and last C within 2 min from B
  • In parallel
    • A and each B within 3 min from A and last C within 4 min from A
• Parameters can be added between events in a pattern
Negations and Aggregates

• Two kinds of negations:
  • Interval based:
    • A and last B
      within 3 min from A
      and not C between B and A
  • Time based:
    • A and not C within 3 min from A

• Similarly, two kinds of aggregates
  • Interval based
    • Use values appearing between two events
  • Time based
    • Use values appearing in a time interval
Iterations

• We believe that explicit operators for repetitions are difficult to use/understand
  • Especially when different selection/consumption policies are allowed
• We achieve the same goal using hierarchies of events
  • Complex events can be used to define (more) complex events
• Recurring schemes of repetitions
  • Macros!
Formal Semantics

• The semantics of each TESLA operator / rule is formally defined through a set of TRIO metric temporal logic formulas
  • They define an equivalence between the presence of a given pattern and the occurrence of one or more complex events, specifying:
    • The occurrence time of complex events
    • The content of complex events
    • The set of consumed events
  • The language is unambiguous
    • A user can in advance check the semantics of her rules
  • This makes it possible to check the correctness of an event detection engine using a model checker
    • Given a history of events and a set of TESLA rules ...
    • ... does the engine satisfy all the formulas?
Abstraction

• Support for uncertainty
  • Many applications deal with imprecise (or even incorrect) data from sources
    • E.g., sensors, RFID, ...
  • Uncertain rules may increase the expressiveness / usefulness of an IFP system

[more on this later]
Abstraction

• Support for QoS policies
  • Allow users to define application-dependent policies
    • Which data is more important?
    • Which items is it better to discard in case of system overload?
    • Which QoS metric is more significant for the application?
      – Completeness of results
      – Latency
      – ...
  • Traditionally offered only by DSMSs
  • Most systems simply adopt a best-effort strategy
Abstraction

• Support for a *Knowledge Base*
  
  • Some systems offer special operations to read from persistent storage
  
  • Possibly a key-feature for the integration with existing DBMSs

• Support for periodic evaluation of rules
  
  • As opposed to purely reactive evaluation
Abstraction

• Capability to dynamically change the set of deployed rules
  • Add / Remove
  • Activate / Deactivate

• Context awareness
  • Tutorial at DEBS 2010
  • Could be used to increase the expressiveness of IFP ...
  • ... but also to simplify / speedup processing
    • Context / Situation used to reason on the status of the system / application
    • Possibly combined with the possibility to activate / deactivate rules
Abstraction

• *Where* to offer the IFP abstraction?
  • As a standalone product/system
    • Similar to DBMSs
  • As a middleware component
    • Similar to a Publish/Subscribe system
    • To guide the interaction of other components
  • As part of a programming language
    • Similar to what LINQ (Language-Integrated Query) offers to C# and to the .Net Framework
    • Explored in EventJava
      – Extension of Java for event correlation
Time

• Many existing systems offer operators that rely on some notion of time
• Main problem: out-of-order arrival of information items
  • Requirements
    • Preserve the semantics of processing
    • Keep the delay for obtaining results as small as possible
  • Solutions
    • Trade-off between correct and low latency processing
      – Sometimes controlled by users through predefined or customizable policies
  • Open issues
    • Solutions for distributed / incremental processing may introduce delay at each processing step
      – Problem not addressed in existing solutions for distributed processing
Processing

• Many processing algorithms proposed
  • DSMSs cannot exploit traditional indexing techniques
    • New effort is required to efficiently evaluate standing queries
  • Most CEP systems are based on automata …
  • … or in general incremental processing
    • Perform processing incrementally, as new information is available
    • Academia: ODE, Cayuga, NextCEP, Amit, TRex, etc.
    • Industry: Esper/Nesper

• We investigated other solutions
  • Keep the whole history of all received information items
    • Ad-hoc data structures that simplify information retrieval
  • Start evaluation only when required
  • Faster!
  • Easier to parallelize
Processing

• Are we using the right algorithms?
  • Can we isolate the best algorithm for a particular set of operators?
    • Accurate analysis of performance
  • Can we switch among different algorithms depending from the workload?
    • Deployed rules ...
    • ... but also analysis of traffic
      – Adaptive middleware
Processing

• Exploit parallel hardware
  • Multi-core CPUs, but also GP-GPUs
  • To handle several rules concurrently
  • To speedup complex operations inside a single rule
    • Only partially investigated
      – E.g., streaming aggregates
    • Can be extended to different operators?
  • Also in this case the best solution may depend from the workload
Our Experience

Processing Time (ms) vs. Number of States

- AIP Multiple Selection
- CDP CPU Multiple Selection
- CDP GPU Multiple Selection
- AIP Single Selection
- CDP CPU Single Selection
- CDP GPU Single Selection

Stream & Complex Event Processing - Models
Our Experience

• GPU can significantly speedup huge computations
  • Fixed overhead to activate it makes it unsuitable for simple tasks
  • Optimal workload: reduced set of very complex rules
  • Additional advantage: CPU cores are free for other task
    • Process simple rules ...
    • ... but also handle marshalling/unmarshalling and communication with sources and sinks

• The trend. GPUs are becoming:
  • Easier to program
  • Suitable for an increasing number of tasks
    • Not only data parallel algorithms
Processing

• Exploit existing models for computation on clusters / clouds
  • Recently received great attention
    • E.g. Map-Reduce
  • Are these models suitable for IFP systems?
    • To efficiently support the expressiveness of existing languages
  • Can we create new models?
    • Explicitly conceived to deal with IFP rules
Distribution of Processing

- Most systems rely on centralized processing
- When distributed processing is allowed, a clustered deployment model is often adopted
  - Cooperating processors are co-located
- The problem of networked distribution has been addressed only marginally
  - Are there applications that involve large scale scenarios?
    - E.g. monitoring applications
  - Potentially introduce new issues
    - Bandwidth may become a bottleneck
    - Data may be transferred over non-dedicated channels
    - Several applications may run on the same “processing network”
Workloads

• A major problem when it comes to evaluate and compare different systems is the lack of workloads

• Benchmark
  • Only one benchmark available: Linear Road
    • Strongly tailored for the Stream model
    • Difficult to adapt to other systems
  • Huge parameter space
    • Difficult to isolate relevant cases
  • DEBS Grand Challenge may be a first step in this direction ...

• Data coming from real / realistic use cases
  • To understand which operators are more important ...
  • ... and what to optimize in the processing algorithm
Security / Privacy

• Define security models and policies
  • Which pieces of information can be consumed by a single operator or a set of operators
  • Important: output data may have a different visibility w.r.t. input data!
    • The system may be allowed to consume single data items, but not to extract new information from them, or to distribute this information

• Implement protocols and algorithms to enforce them
Thanks for your Attention!
Questions?