ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Complex Event Processing and a Case Study

Federico Chesani

Department of
Computer Science and Engineering
University of Bologna

The Internet of Thing paradigm, Big Data, and other…

We are assisting to an exponential increase of:

- "simple" devices that have sensors (e.g., traffic lights presence sensors)
- "simple" devices that have a net connection, and whose data can be accessed continuously at run-time (e.g., environmental and weather stations are queried through the net at runtime)
- "complex devices" that continuously provides a flow of information (security cams, webcams, etc.)

All these devices are producing a huge quantity of data, thus giving us the "Big Data" challenge.

The Internet of Thing paradigm, Big Data, and other…

The Big Data challenges…

- collecting huge amount of information
- storing huge amount of information
- filter, analyze, and derive new knowledge from this huge amount of information, in a timely manner
- decide when it would be appropriate to delete huge amount of information
- protecting users privacy and rights about this huge amount of information

Applications can be the most various and disparate… my personal feeling is that

- marketing needs are driving and pushing the research, in this moment…
- … strictly followed by business ICT frameworks (big ICT players), for supporting industrial applications (e.g., industry 4.0)

The Complex Event Processing paradigm…

Complex Event Processing (CEP) is a paradigm (older than IoT and Big Data), for dealing with such huge amount of information

- Collected information is described in terms of **events**
  - A **description** of something, in some (logic?) language
  - A **temporal** information, about **when** something happened.
  - Events can be **instantaneous**, or can have a **duration** (philosophical/ontological issue, but with a number of practical consequences)

It is a very natural, human-like way of describing the information collected by sensors…

- Underneath, it is adopting the human reasoning way of abstracting from correlation to causality, with the flow of the time

The Complex Event Processing paradigm…

Complex Event Processing (CEP) is a paradigm (older than IoT and Big Data), for dealing with such huge amount of information

Problem: events alone usually do not provide enough information

Example: A cross road has been added a (buried in the ground) sensor that is able to detect if a car is passing over the sensor itself.

The outcome is something like:

```
1528038740909 something detected
1528038779384 something detected
1528038790093 something detected
1528038805045 something detected
…
```

So, what?

The Complex Event Processing paradigm…

Problem: events alone usually do not provide enough information

Example: A cross road has been added a (buried in the ground) sensor that is able to detect if a car is passing over the sensor itself.

The outcome is something like:

1528038740909 something detected

1528038779384 something detected

1528038790093 something detected

1528038805045 something detected

…

In the cross road, we are interested to understand if the road is jammed of traffic or not…

… we would need to extract from the single (simple) events such information…

… for example, by simply **averaging** the number of events within a **sliding temporal window**…

The Complex Event Processing paradigm…

Problem: Maritime Surveillance (from works of Artikis and colleagues)

For each ship in the world, events are collected representing:

- id of the vessel
- GPS location
- time of the detection

Security questions:

- Fast approaching: a vessel is moving at high speed towards other vessels
- Suspicious delay: a vessel fails to report its position, and the estimated speed during the information gap is estimated as low
- Possible rendez-vous: two vessels are suspiciously delayed, at the same time, in the same location

The Complex Event Processing paradigm…

Complex Event Processing is about dealing with simple events, reason upon them and derive complex events.

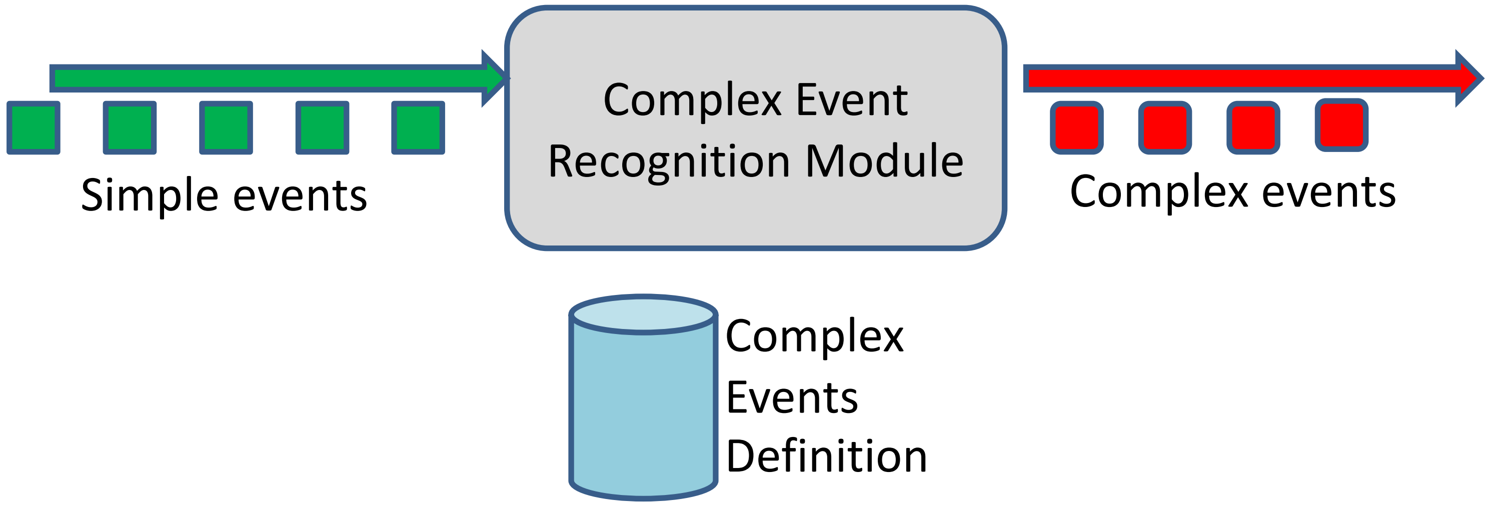
- **Simple events**: events detected by the system underneath, their information payload alone does not provide any immediate answer to our questions
- **Complex events**: events generated by the system itself (they do not necessarily maps on real events), that provide higher informative payload, and that can partially/totally answer our application questions

Simple and complex do not refer to the information payload, but rather to their ability to provide answers to the application questions.

The Complex Event Processing paradigm…

Complex Event Processing is about dealing with simple events, reason upon them and derive complex events.

The term CEP capture the whole framework needed to recognize events. Usually, authors refer also to **Complex Event Recognition Languages**.



The Complex Event Processing paradigm…

According to Artikis et al., CEP/CER can be characterized by:

INPUT:

- Instantaneous events.
- Durative events.
- Context information.

OUTPUT: durative events

- The interval may be open.
- Relational & non-relational events.
- Limits or No limits on the temporal distance between the events comprising the composite activity (support to sliding windows of possibly infinite length).
- Sequence constraints.
- Concurrency constraints.
- Spatial reasoning.
- Event hierarchies.
- Different sensors involved (different meaning) (Sensor Fusion)

The Complex Event Processing paradigm…

A number of different approaches have been proposed for applying CEP.

A classification, from the viewpoint of the adopted framework:

- Automata-based (e.g., Regular Expression)
- Tree-based
- Logic-based

In the following we will focus on Logic-based approaches, and in particular, on the Event Calculus logical framework.

CEP and Drools

Drools provide a support for doing CEP: **Drools Fusion**.

- Events are particular facts
- Events have a timestamp (but support for interval-time events as well)
- Support to Allen algebra for querying instantaneous events with respect to time interval

- On the base of the defined rules, it automatically decides the discard of events:
    - It avoids working memory cluttering and all related problems
    - It requires particular attention from the user side: no writing of rules that would imply to connect events too far in the time line
    - It supports aggregation operators
    - Sensor Fusion is not supported as operators, but can be achieved through the writing of chained rules and precise domain semantics
    - It supports the negation operator over time windows
    - It support fixed as well as sliding windows

CEP and Drools

Few basic assumptions (from the Drools manual):

- Usually required to process **huge volumes of events**, but only a small percentage of the events are of real interest.
- Events are usually **immutable**, since they are a record of state change.
- Usually the rules and queries on events must run in **reactive** modes, i.e., react to the detection of event patterns.
- Usually there are strong **temporal relationships** between related events.
- Individual events are usually not important. The system is concerned about **patterns of related events** and their relationships.
- Usually, the system is required to perform **composition** and **aggregation** of events.

CEP and Drools – negation: different behavior

Drools Fusion adopts a view of the flow of events based on Stream: events come in with a timestamp…

Example:

```
rule "Sound the alarm”
when
        $f : FireDetected( )
        not( SprinklerActivated( ) )
then
        // sound the alarm
end
```

When the not is evaluated? Different behavior between Cloud mode and Stream mode…

CEP and Drools – negation: different behavior

What about **not observing events**?

Example:

```
rule "Sound the alarm”
when
      $f : FireDetected( )
      not( SprinklerActivated( this after[0s,10s] $f ) )
then
      // sound the alarm
end
```

When the not is evaluated? Different behavior between Cloud mode and Stream mode…

Note: reasoning over temporal intervals requires the notion of a **clock**…

CEP and Drools – Sliding Windows

What about focusing on the events that happened only the last 24 hours?
How to focus on the last 100 events only?

“**Sliding Windows** are a way to scope the events of interest by defining a window that is constantly moving.”

Two types of sliding windows are supported:

a) time-based windows
b) length-based windows

CEP and Drools – Time-based sliding windows

"**Sliding Windows** are a way to scope the events of interest by defining a window that is constantly moving."

Two types of sliding windows are supported:

a) time-based windows

b) length-based windows

```
rule "Sound the alarm in case temperature rises above threshold”
when
        TemperatureThreshold( $max : max )
        Number( doubleValue > $max ) from accumulate(
           SensorReading($temp : temperature ) over window:time(10m ),
           average( $temp )
        )
then
        // sound the alarm
end
```

CEP and Drools – Sliding Windows

"**Sliding Windows** are a way to scope the events of interest by defining a window that is constantly moving."

Two types of sliding windows are supported:

a) time-based windows

b) length-based windows

```
rule "Sound the alarm in case temperature rises above threshold"
when
        TemperatureThreshold( $max : max )
        Number( doubleValue > $max ) from accumulate(
SensorReading($temp : temperature ) over window:length(10 ),
           average( $temp )
        )
then
        // sound the alarm
end
```

CEP and Drools – Events Expiration

Due to the huge amount of events that should be treated, events will be discarded from the working memory ASAP…

Two ways for determining the expiration:

Explicit expiration:

```
declare StockTick
        @expires( 30m )
end
```

Implicit expiration:

```
rule "correlate orders”
when
        $bo : BuyOrderEvent( $id : id )
        $ae : AckEvent( id == $id, this after[0,10s] $bo )
then
        // do something
end
```

CEP and Drools – Temporal Reasoning

A number of Allen temporal operators are supported:

After and **Before**

```
$eventA : EventA( this after[ 3m30s, 4m ] $eventB )
```

meaning:

```
3m30s <= $eventA.startTimestamp - $eventB.endTimeStamp <= 4m
```

```
$eventA : EventA( this before[ 3m30s, 4m ] $eventB )
```

meaning:

```
3m30s <= $eventB.startTimestamp - $eventA.endTimeStamp <= 4m
```

CEP and Drools – Temporal Reasoning

A number of Allen temporal operators are supported:

Coincides:

```
$eventA : EventA( this coincides $eventB )
```

with threshold:

```
$eventA : EventA( this coincides[15s, 10s] $eventB )
```

During:

```
$eventA : EventA( this during $eventB )
```

meaning:

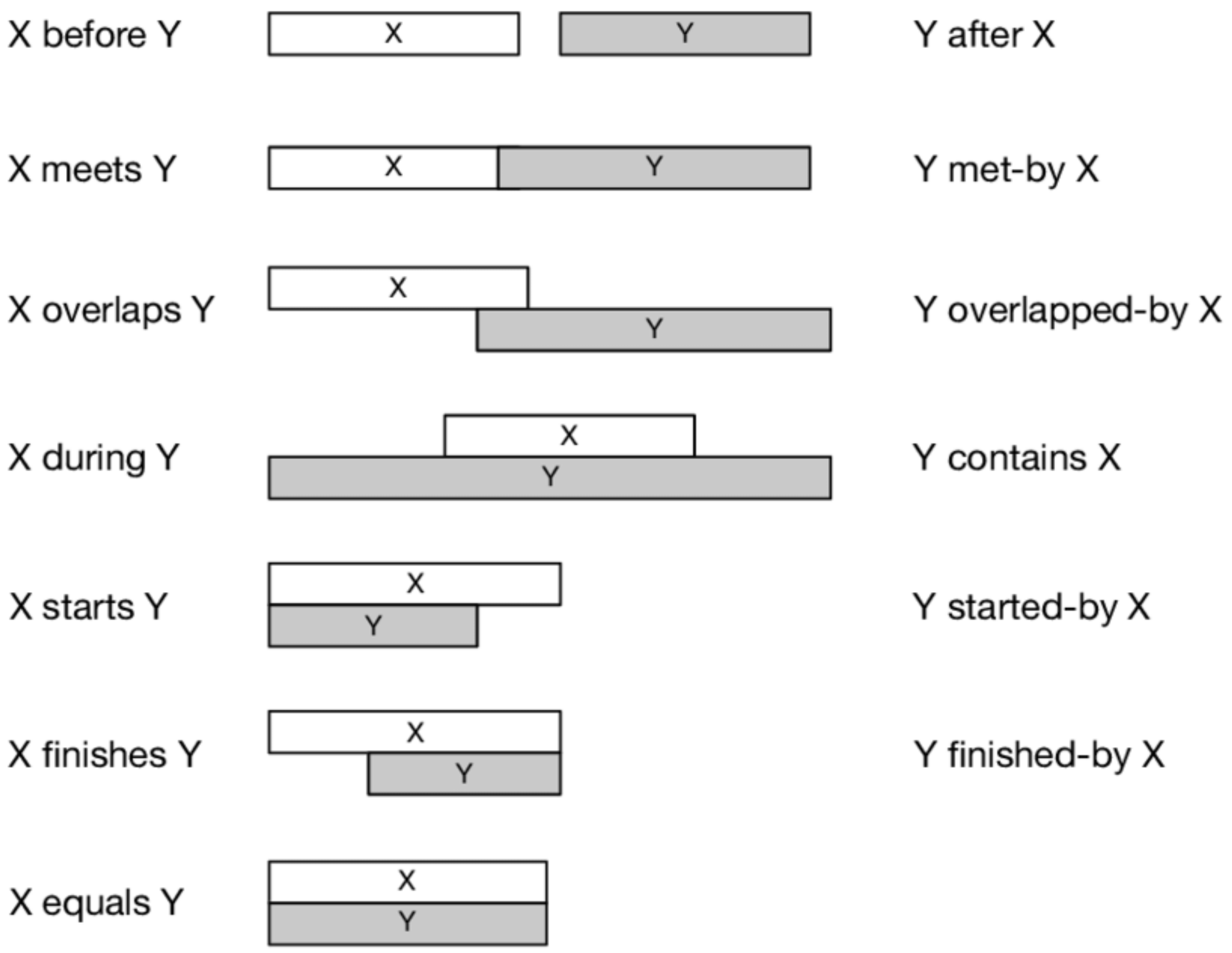
```
$eventB.startTimestamp < $eventA.startTimestamp <=
$eventA.endTimestamp < $eventB.endTimestamp
```

CEP and Drools – Temporal Reasoning

A number of Allen temporal operators are supported:

- **finishes**
- **finishedby**
- **includes**
- **meets**
- **metby**
- **overlaps**
- **overlappedby**
- **starts**
- **startedby**

CEP and Drools – Allen operators brief recall



Taken from: Colonius, Immo. (2015). Qualitative Process Analysis : Theoretical Requirements and Practical Implementation in Naval Domain.

CEP and Drools – Reasoning about the state of the system

CEP is a very nice way for reasoning about events… but…
…more complex systems require to reason about the state of a system.

How to, in Drools?
Let us use facts (and not events), to represent the state of the system

- Fast and easy solution for simple state representation
- Ok if only a type of event affects a state property

Adopt the **Event Calculus Framework** (Kowalski and Sergot, 1986)

- More complex solution, but cleaner
- Allows to link multiple different events to the same state property
- State property changes can depend also from other state
- Allows to reason on meta-event of state property change (clip and de-clip meta-events)

Reactive Event Calculus and Drools

- Drools already overcomes a number of issues due to logic approaches
- EC provides any way a structured and clean way for addressing the representation of the state of a system

Take Home message: When needed…

- **Cleanly separate Events from state properties (fluents)**
- **Determine which are the events that affect a fluent (i.e., they change its truth)**
- **Define a rule for any of these events, the consequence of the rule will be a special UpEvent (one for each fluent)**
    - **These rules con combine a number of conditions**
- **Define a rule that as a consequence of the UpEvent, will trigger the fluent to true**
- **Same for DownEvents**
- **Use UpEvents and DownEvents for meta-reasoning**
- **If possible, remove fluents from the WM, so as to avoid memory cluttering**

Reactive Event Calculus and Drools – Case Study

The HABITAT Project

- Financed by the Region Emilia-Romagna for the technology transfer to middle size industries, companies, firms
- Two year project, ended the 31 of July 2018
- Main goal: develop an eco-system of IoT with sensors, so as to implement smart homes that will ensure people a better living
- Target: Elderly, but still independent; Elderly, already assisted by caregivers

- Radar for indoor positioning
- Sensorized armchair, with posture evaluation
- Sensorized belt, for physical activity recording and analysis
- A Radio, as an output console (big tablet with apps designed for elderly)
- Apps running on smartphones/smartwatch

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Reactive Event Calculus and Drools – Case Study

Scenario

- Some parts of your flat are to be intended as red zones. For example, the area around the exit door.
- If the elder is alone, and he is trying to get out, send him a reminder of the things she/he should do before exiting (remember the keys, switch off the stove, close the windows)
- If the elder is attended by a caregiver, alarm the caregiver that she/he is trying to get out

- Implicit: elderly living alone are autonomous, elderly with a caregiver are not autonomous anymore

The following solutions have been developed in collaboration with Dr. Daniela Loreti.

Reactive Event Calculus and Drools – Case Study

Scenario

- Some parts of your flat are to be intended as red zones. For example, the area around the exit door.
- If the elder is alone, and he is trying to get out, send him a reminder of the things she/he should do before exiting (remember the keys, switch off the stove, close the windows)
- If the elder is attended by a caregiver, alarm the caregiver that she/he is trying to get out
- Keep track if the caregiver answer the alarm, or not…

- Which objects? Which fluents? What if the caregiver does not answer?

Reactive Event Calculus and Drools – Case Study

Scenario – Internal events (meta events)

```
/*
 * A user has been detected to be in red zone
 */
declare UserRedZoneUpEvent
        @propertyReactive
        @role(event)
        @timestamp(timestamp)
        user : User
        caregiver : User
        room : String
        since : Long
        timestamp : Long
end
```

```
/*
 * The red zone situation has been solved
 */
declare UserRedZoneDownEvent
        @propertyReactive
        @role(event)
        @timestamp(timestamp)
        user : User
        timestamp : Long
end
```

Reactive Event Calculus and Drools – Case Study

Scenario – Fluents

```
/*
 * The fluent represent the state of the Notification
 */
declare UserRedZoneNotificationFl
        notification : NotificationMessage
        user : User
        room : String
        since : Long
        whenRaised : Long
        whenAcknowledged : Long
        whenIgnored : Long
end
```

Reactive Event Calculus and Drools – Case Study

Scenario – Rule detecting the RedZone

```
rule "Detect the user red zone condition upon LocationEvents - caregiver not
any information"
when
        $posEv : PositionEvent ( $user : user, $room : locationURI,
$timestamp : timestamp ) from entry-point "EventStream"
        $posFl : UserInRoomFluent (user == $user ,
$user.getRole()==User.Role.SUBJECT ,  room == $room, $room memberOf
$user.getRedZones(), $since : this.getSince().getTime(), $timestamp - $since
> $user.getTriggerTimeForRedZone($room) )
        not ( $fl : UserRedZoneNotificationFl ( $user == user , whenRaised <
$timestamp , (whenAcknowledged==null || whenAcknowledged>$timestamp),
(whenIgnored==null || whenIgnored>$timestamp) ))
        not ( $evDown : UserRedZoneDownEvent ($user == user, this before
[0s,20s] $posEv) )
        not ($cg : CaregiverPresenceFl( ) )
then
        UserRedZoneUpEvent userRedZoneUpEvent = new UserRedZoneUpEvent(
$user , null, $room, $since, System.currentTimeMillis() );
        insert(userRedZoneUpEvent);
end
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Reactive Event Calculus and Drools – Case Study

Scenario – Rule detecting the RedZone

```
rule "Detect the user red zone condition upon LocationEvents - caregiver
absent"
when
        $posEv : PositionEvent ( $user : user, $room : locationURI,
$timestamp : timestamp ) from entry-point "EventStream"
        $posFl : UserInRoomFluent (user == $user ,
$user.getRole()==User.Role.SUBJECT ,  room == $room, $room memberOf
$user.getRedZones(), $since : this.getSince().getTime(), $timestamp - $since
> $user.getTriggerTimeForRedZone($room) )
        not ( $fl : UserRedZoneNotificationFl ( $user == user , whenRaised <
$timestamp , (whenAcknowledged==null || whenAcknowledged>$timestamp),
(whenIgnored==null || whenIgnored>$timestamp) ))
        not ( $evDown : UserRedZoneDownEvent ($user == user, this before
[0s,20s] $posEv) )
        $cg : CaregiverPresenceFl( presence == false , since < $timestamp )
then
        UserRedZoneUpEvent userRedZoneUpEvent = new UserRedZoneUpEvent(
$user , null, $room, $since, System.currentTimeMillis() );
        insert(userRedZoneUpEvent);
end
```

Reactive Event Calculus and Drools – Case Study

Scenario – Rule detecting the RedZone

```
rule "Detect the user red zone condition upon LocationEvents - caregiver
present"
when
        $posEv : PositionEvent ( $user : user, $room : locationURI,
$timestamp : timestamp ) from entry-point "EventStream"
        $posFl : UserInRoomFluent (user == $user ,
$user.getRole()==User.Role.SUBJECT ,  room == $room, $room memberOf
$user.getRedZones(), $since : this.getSince().getTime(), $timestamp - $since
> $user.getTriggerTimeForRedZone($room) )
        not ( $fl : UserRedZoneNotificationFl ( $user == user , whenRaised <
$timestamp , (whenAcknowledged==null || whenAcknowledged>$timestamp),
(whenIgnored==null || whenIgnored>$timestamp) ))
        not ( $evDown : UserRedZoneDownEvent ($user == user, this before
[0s,20s] $posEv) )
        $cg : CaregiverPresenceFl( presence == true , since < $timestamp )
then
        UserRedZoneUpEvent userRedZoneUpEvent = new UserRedZoneUpEvent(
$user , $cg.getCaregiver() , $room, $since, System.currentTimeMillis() );
        insert(userRedZoneUpEvent);
end
```

Reactive Event Calculus and Drools – Case Study

Scenario – Generating the notification

```
rule "generate the USER_RED_ZONE Notification"
when
        $ev: UserRedZoneUpEvent ( $user : user, $caregiver : caregiver,
$room : room, $since : since)
then
{
        if ($caregiver == null) {
                // send custom message to the user
        }
        else {
                // notify the caregiver, and then
                insert ( new UserRedZoneNotificationFl ( $notification,
$user, $room, $since, System.currentTimeMillis(), null, null) );
        }
end
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Reactive Event Calculus and Drools – Case Study

Scenario – Managing the Ok answer

```
rule "UserRedZone Answer: OK"
when
        $ans : AnswerMessage ( answer == AnswerMessage.ANSWER.YES,
$idRefMessage : idRefMessage, $timestamp : timestamp )  from entry-point
"EventStream"
        $fl :  UserRedZoneNotificationFl ( notification.getIdSEPA() ==
$idRefMessage , whenRaised < $timestamp , whenAcknowledged==null ,
whenIgnored == null , $user : user)
then
        $fl.setWhenAcknowledged(System.currentTimeMillis());
        UserRedZoneDownEvent de = new UserRedZoneDownEvent( $user,
System.currentTimeMillis());
        insert (de);
end
```

Reactive Event Calculus and Drools – Case Study

Scenario – Managing the “not any” answer

```
rule "UserRedZone Answer: NOT ANY"
no-loop
when
    $evUp : UserRedZoneUpEvent ( $user : user, $room : room, $since : since)
    $fl :  UserRedZoneNotificationFl ( $notification: notification,
whenAcknowledged==null , whenIgnored == null , $user == user, $room == room)
    not $ans : AnswerMessage ( idRefMessage == $notification.getIdSEPA(),
$timestamp : timestamp, this after[0s, 30s] $evUp )  from entry-point
"EventStream"
then
         $fl.setWhenIgnored(System.currentTimeMillis());
         UserRedZoneDownEvent de = new UserRedZoneDownEvent( $user,
System.currentTimeMillis());
         insert (de);
end
```

Reactive Event Calculus and Drools – Case Study

Scenario – Closing the notification fluent

```
rule "UserRedZone Closing the advice fluent"
no-loop
when
        $ev : UserRedZoneDownEvent( $user : user, $timestamp : timestamp)
        $fl : UserRedZoneNotificationFl ( user == $user , whenRaised <
$timestamp)
then
        //delete ( $ev );
end
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Federico Chesani

DISI – Department of Computer Science and Engineering

federico.chesani@unibo.it