



Towards a Flexible Data Stream Analytics Platform based on the GCM Autonomous Software Component Technology



Membre de UNIVERSITÉ CÔTE D'AZUR



Prof Françoise Baude, Léa El Beze (MSc), Miguel Oliva (MSc)

CNRS I3S UMR 7271,
University of Nice Sophia-Antipolis
France

Contact: baude@unice.fr

Agenda



I. Motivation and Requirements: flexible



analytics for big data streams

II. Background technology: GCM/ProActive

III. Proposition: GCM Streaming platform

IV. Current work & conclusion

Big data stream analytics: flexibility

- In-memory, on-line efficient analytics on big data volume
 - From IoT, Social networks, etc
 - Situation detection: Fraud, natural disaster
 - Crisis prevention & management, urgent computing...
 - Not all the situations must be handled the same way
 - Normal versus exceptional
 - Each situation requires a variant of the analytics
 - Program the automatic switch between analytics
- ➔ Flexible, self-adaptable analytics : Minimize STOP-(UN)DEPLOY-START effects
- Programming & support of the « core » analytics
 - Extension to **runtime** reconfiguration rules
-

Big data Stream analytics platforms:

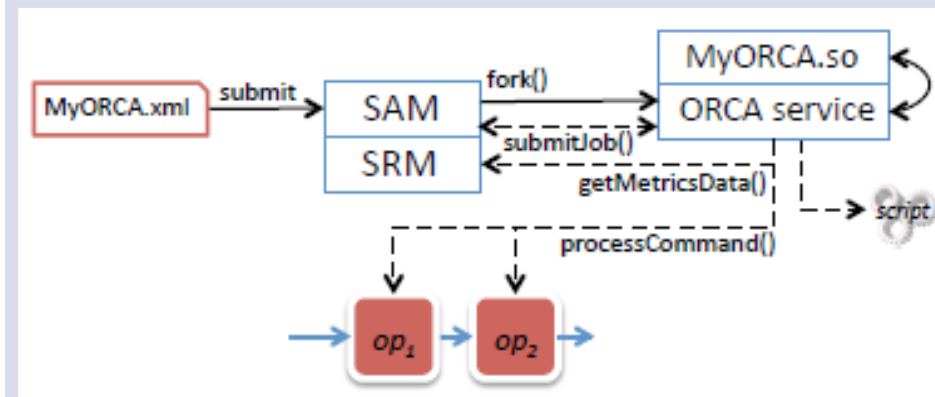
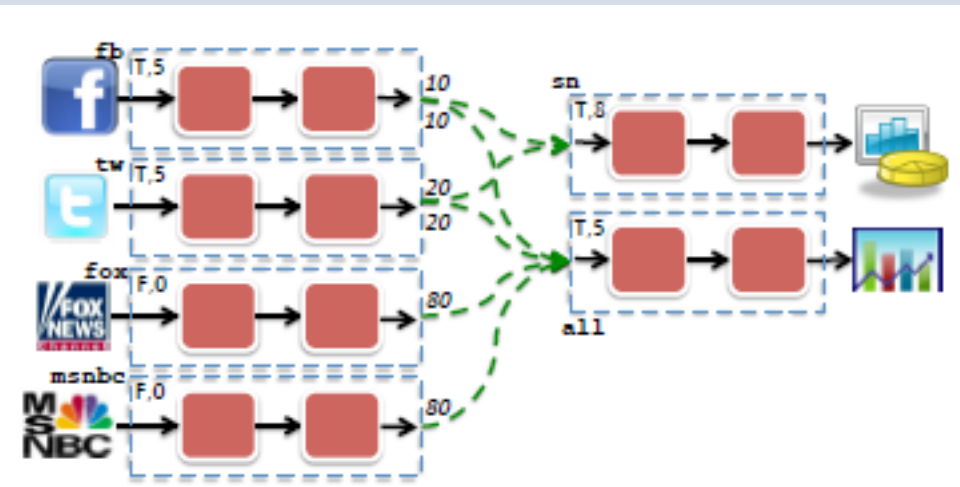
Comparison criteria

- Reconfiguration capability of the Dataflow Analytics graph ?
 - Non functional (e.g. to not violate SLAs)
 - Functional (to adapt the analytics)
- Reconfigurability is expected to be more easy to handle if
 - Data flow graph close to runtime view
 - Data flow graph operators
 - support elasticity / intra parallelism
 - can be replaced on the fly
 - Language/protocol for reconfiguration rules/program:
 - Autonomic support
 - Highly expressive
 - Handling of tuples under analysis: expressive and safe
 - Control /Enactment of the process: localized & distributed

Inspiring platforms / ideas (1/2)

- IBM SPL/ Infosphere Streams « ecosystem »
 - Higher order composite coarse-grained operators (SPL)
 - Separate adaptation logic to control **when** to submit / cancel **new** applications

G. Jacques-Silva, B. Gedik, R. Wagle, K.-L. Wu, and V. Kumar, “Building user-defined runtime adaptation routines for stream processing applications,” VLDB Endowment, 2012



Arc annotations indicate how long the source application must run before the target one can start.

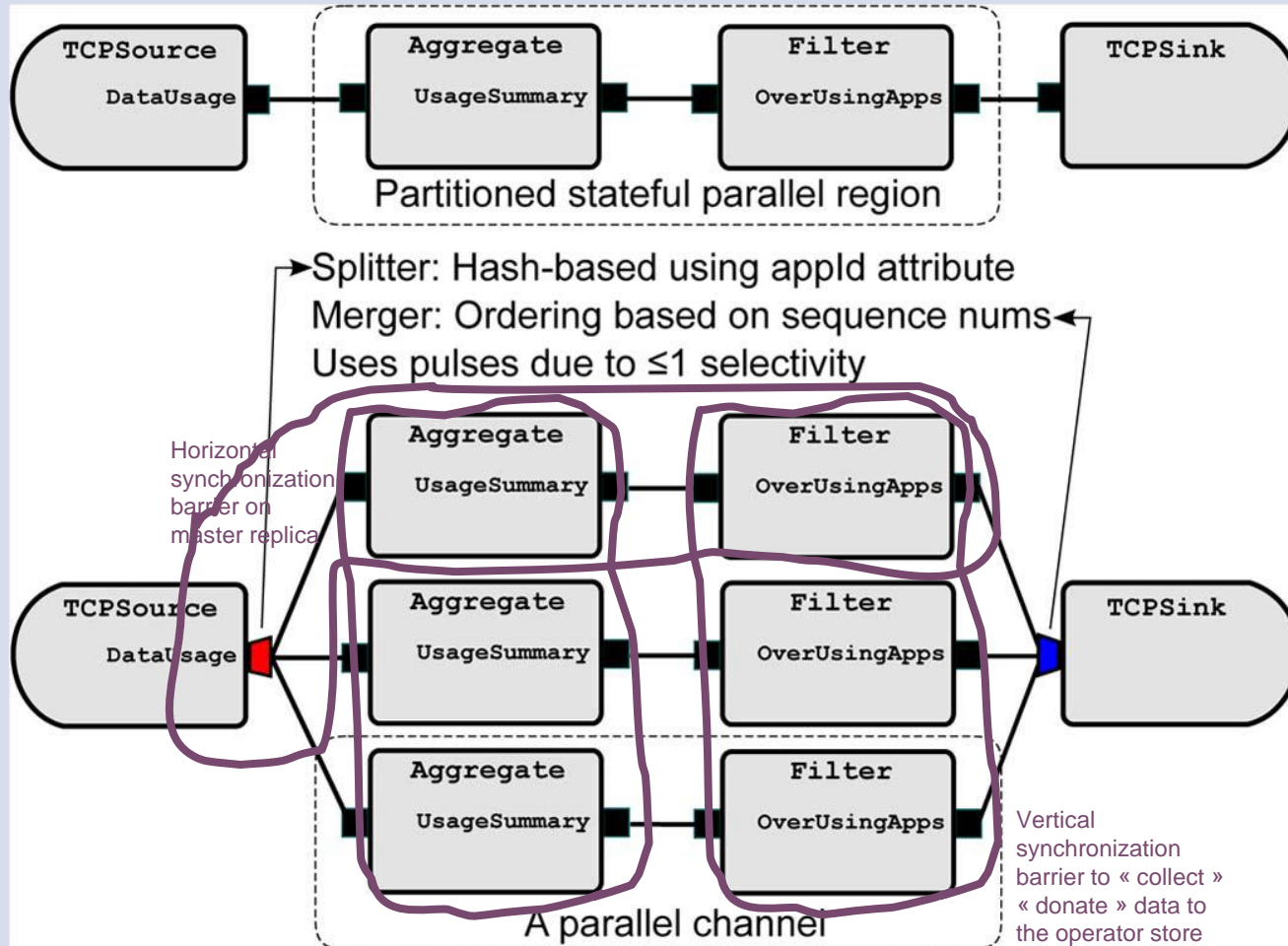
Orchestration of the adaptation is done centrally

Inspiring platforms / ideas (2/2)

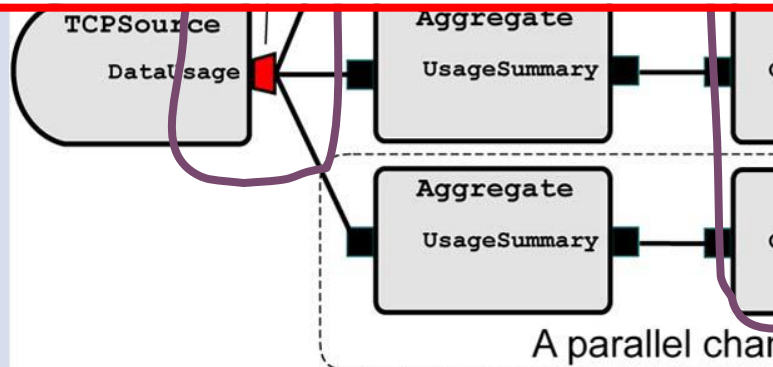
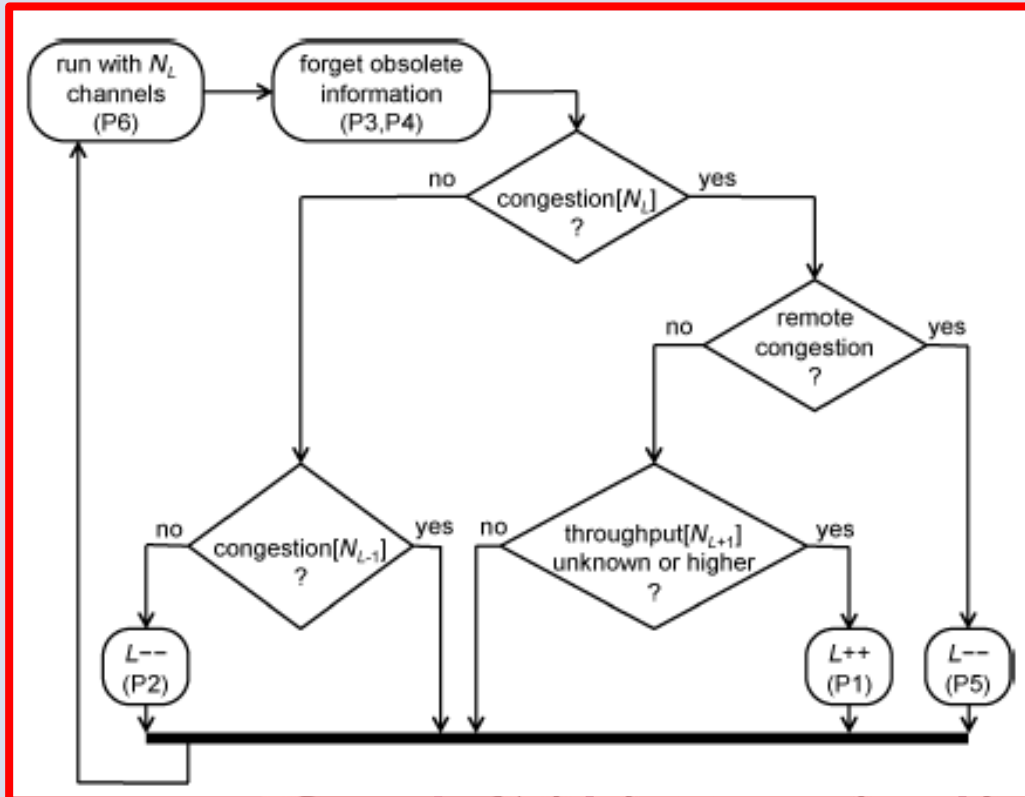
- Auto-parallelization of operators :
 - Classical *autonomic elasticity* based on CPU/network loads
 - Operator Fission and Contraction
 - Application-driven triggers definition
 - ▢ E.g.: millions of tweets 2013 FIFA Confederations Cup + application that calculates public sentiment changes during soccer matches.
- Goal: to meet SLA = “x sec/ analyzed tweet”, **anticipate burst** of tweets as high changes in sentiments occur

A. A. D. Souza and M. A. Netto, “Using application data for SLA-aware auto-scaling in cloud environments”
IEEE MASCOTS, 2015

Ex: Operator Fission & state migration protocol



tuple management migration



Algorithm 2: Update of the number of channels.

Require: T : current throughput, C : current congestion
 procedure *getNumberOfChannels*(T, C)

/(P3) and (P4): congestion and throughput adapt*/*

$l_c \leftarrow \text{checkLoadChangeViaCongestion}(C)$

$l_t \leftarrow \text{checkLoadChangeViaThroughput}(T)$

if $l_c = \text{LessLoad}$ or $l_t = \text{LessLoad}$

$\forall i \in [0 \dots L] C_i \leftarrow \text{false}; T_i^+ \leftarrow 0$

if $l_c = \text{MoreLoad}$ or $l_t = \text{MoreLoad}$

$\forall i \in (L \dots L^*) C_i \leftarrow \text{true}; T_i^+ \leftarrow \infty$

*/*update info on current level*/*

$P_L \leftarrow P; P \leftarrow P + 1$

$T_L^+ \leftarrow T; C_L \leftarrow C$

if $T_L^+ = \text{nan}$ then $T_L^+ \leftarrow T$

/ update the current level */*

$r \leftarrow (P_{L-1} = P_L - 1) \text{ and } C_{L-1} \text{ and } C_L \text{ and } T_L^+ \leq T_{L-1}^+$

if r */* (P5): remote congestion */*

$T_{L-1}^+ \leftarrow \text{nan}; L \leftarrow L - 1$

else if C */* (P1): expand */*

if $L < L^* - 1$ and $T_{L+1}^+ \geq T$

$T_{L+1}^+ \leftarrow \text{nan}; L \leftarrow L + 1$

else */* (P2): contract */*

if $L > 0$ and $\neg C_{L-1}$

$T_{L-1}^+ \leftarrow \text{nan}; L \leftarrow L - 1$

return N_L */* (P6): rapid scaling */*

Analysis of state of the art: outcome

- Open questions are mostly software-engineering related:
 - Which language(s) to design self-adaptation algorithms
 - Clear and clean separation of concerns
 - Generic still customizable rules taking also Functional concerns in account
 - (self-)Evolvable rules (?)
 - Global distributed architecture (distributed controllers)
- Proposition towards solving these open questions:
 - Select an appropriate distributed software technology

➤ **Grid Component Model (GCM)** *Core GRID*

An extension of Fractal for Distributed computing

➤ **GCM/ProActive**

An implementation of GCM using Multi-Active objects

Agenda

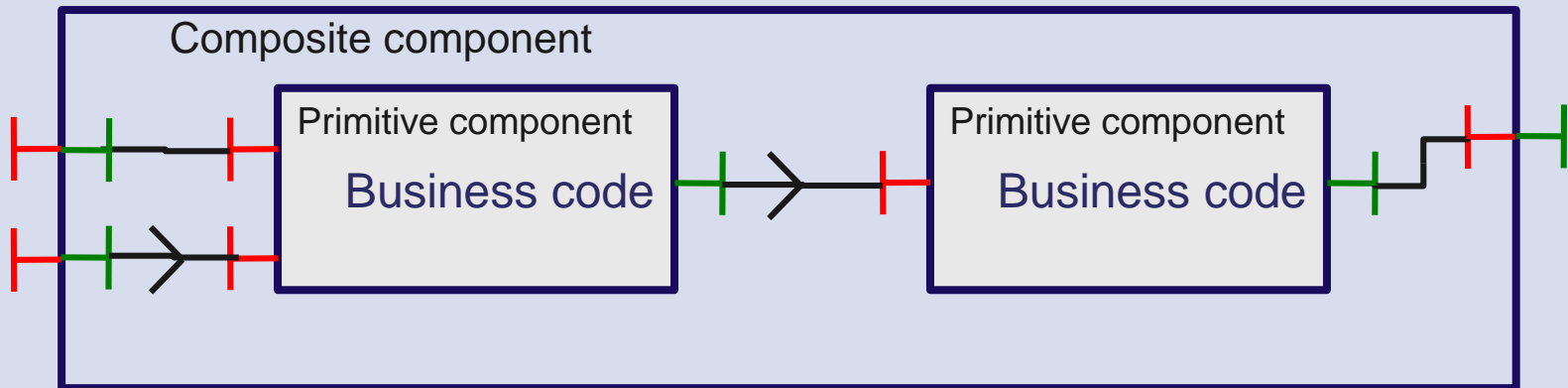
**I. Motivation and Requirements: flexible
analytics for big data streams**

 **II. Background technology: GCM/ProActive** 

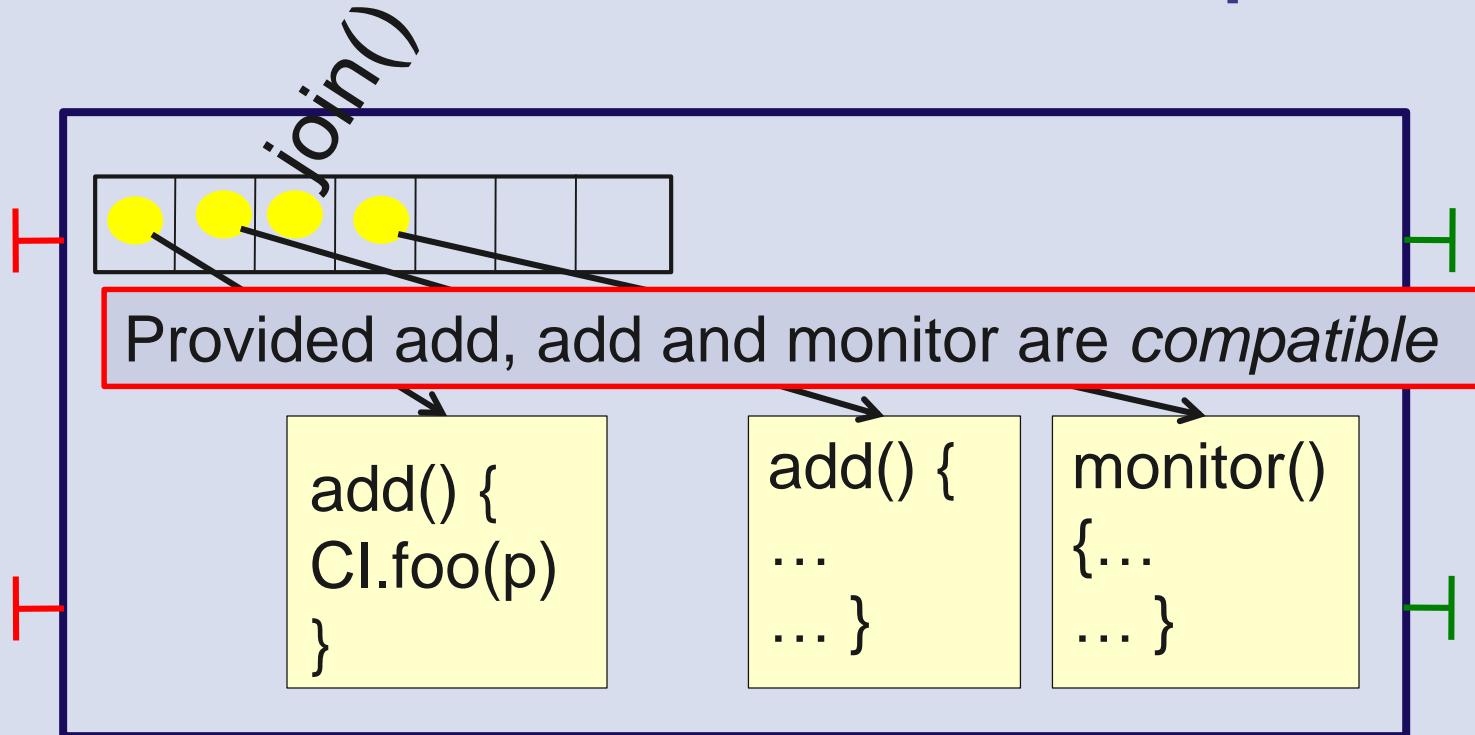
III. Proposition: GCM Streaming platform

IV. Current work & conclusion

What is a GCM component





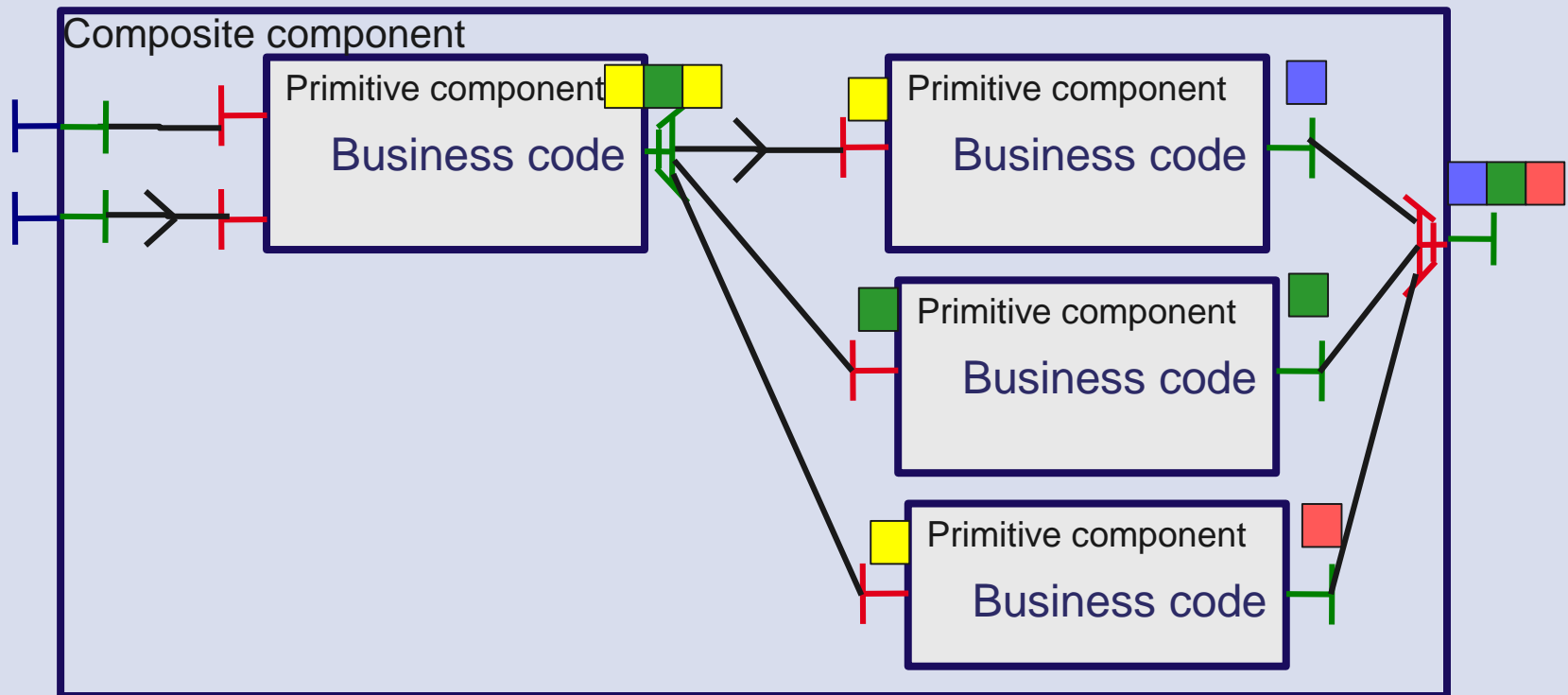
Multi-Active GCM/ProActive Component



Note: *monitor* is compatible with join

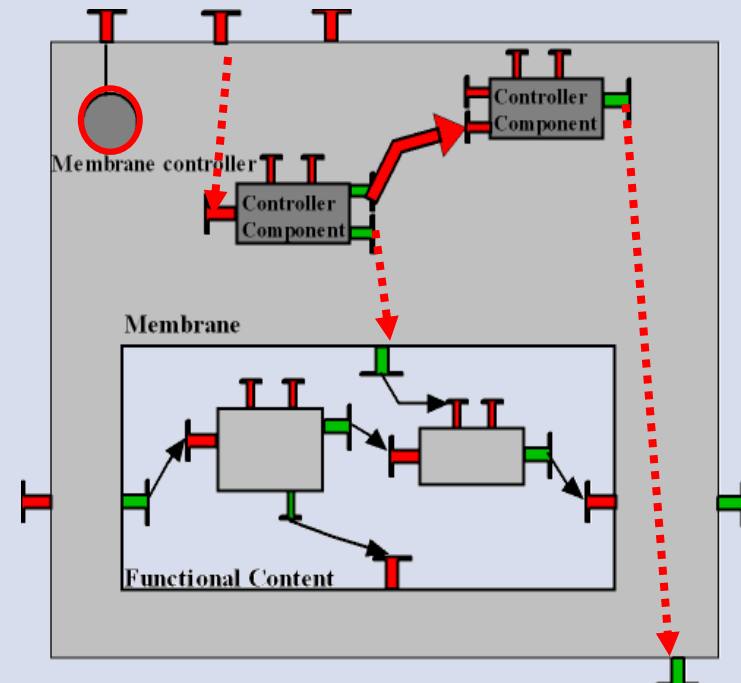
Collective interfaces (GCM)

- One-to-many = multicast 
- Many-to-one = gathercast 
- Distribution and synchronisation/collection policies for invocation and results



Separation of concerns in GCM architecture

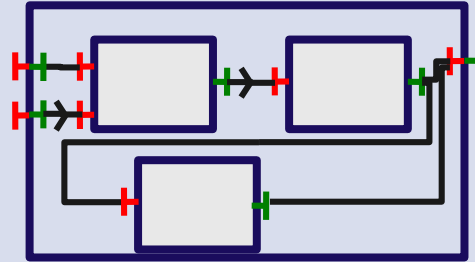
- **Content**: responsible for business logic
- **Membrane**: responsible for control part
- Functional and non-functional **interfaces**
- Business logic and control part can be **designed separately**



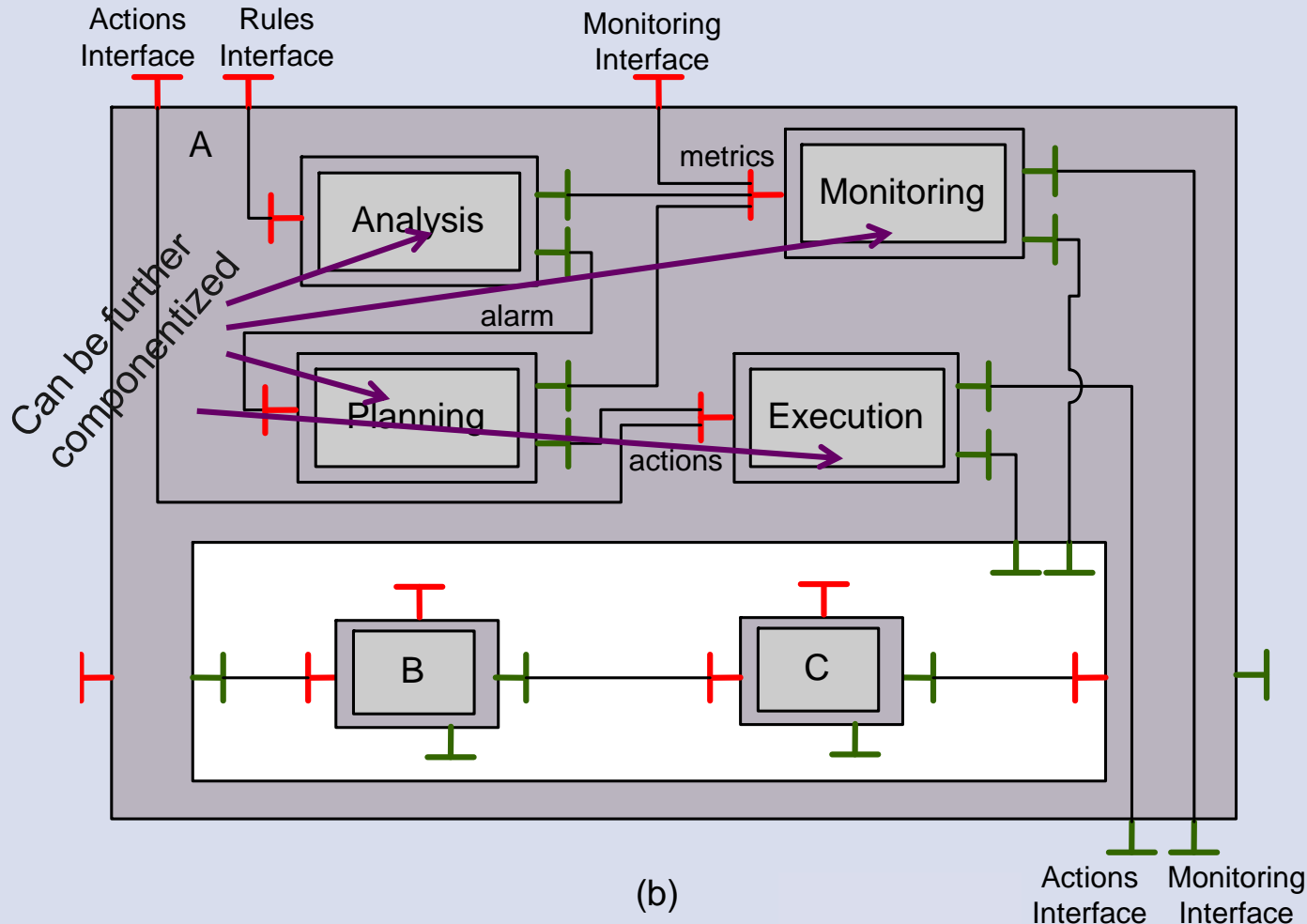
Then aggregated when describing the complete Architecture (XML-based Architecture Description Language)

Adaptation in the GCM

- Functional adaptation: adapt the architecture + behaviour of the application to new requirements/objectives/environment
- Non-functional adaptation: adapt the architecture of the container+middleware to changing environment/NF requirements (QoS ...)
 - May impact the application also
- Both functional and non-functional adaptation expressed as reconfigurations:
 - sets of GCM architecture transformation operations
 - Programmed using GCM Script or direct GCM API calls



Autonomic components: MAPE loop in GCM/ProActive

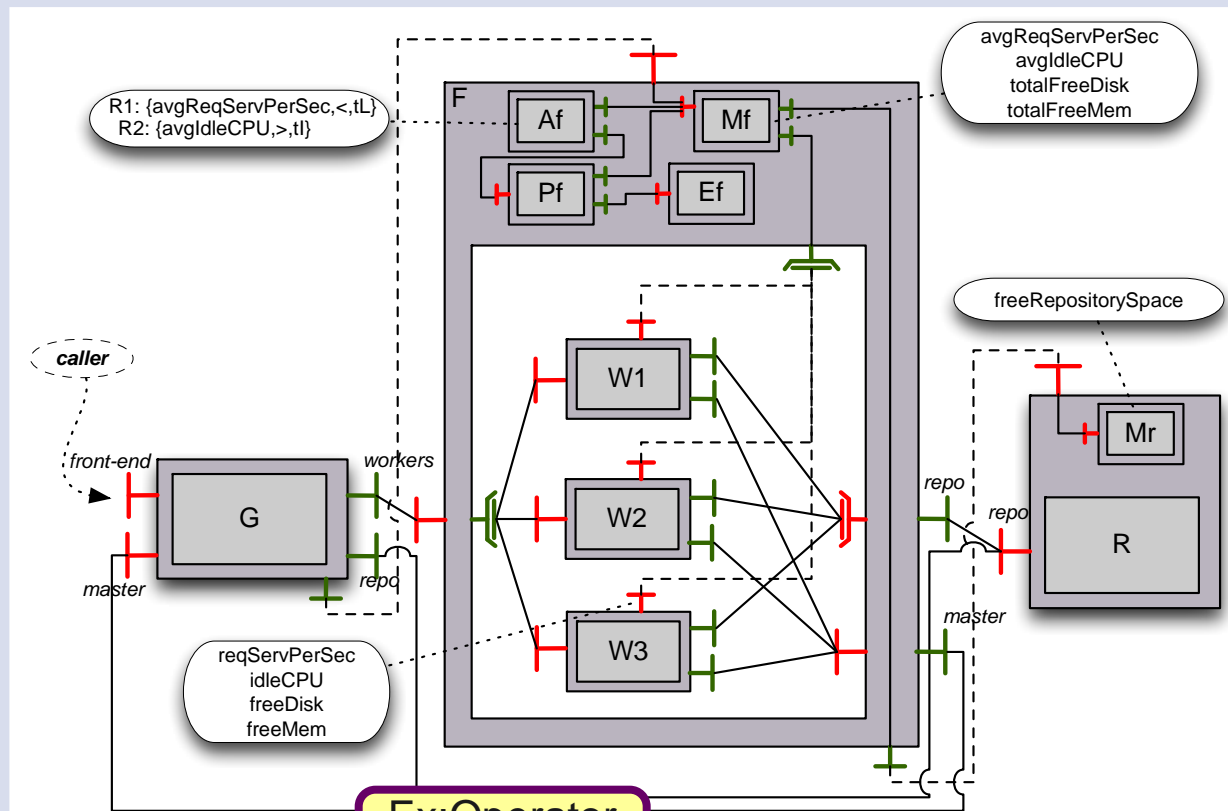


MAPE API:

- Allow runtime adaptation of the MAPE rules
- Can be under responsibility of controller components

Functional sensors possible

Typical usecase – Adaptive Farm pattern



Ex: Operator
fission

```

action addWorker() {
  $W4 = newComponent("worker.ad"),
  add($F,$W4);
  bind($F/interface::wi,$W4/interface::wi)
  bind($F/interface::wo,$W4/interface::wo)
  bind($F/interface::wm,$W4/interfaces::master)
  bind($F/interface::intmetrics,$W4/interface::metrics)
}
    
```

```

action changeRepository(RP) {
  unbind($F/interface::repo);
  unbind($F/interface::extmetrics);
  unbind($G/interface::repo);
  bind($F/interface::extmetrics,$RP/interface::metrics)
  bind($F/interface::repo,$RP/interface::repo)
  bind($G/interface::repo,$RP/interface::repo)
}
    
```

(a)

(b)

Deployment on distributed infrastructures



- Any GCM application can be deployed on a set of computing nodes, as a simple **Java** application
- Using the Grid/Cloud configurable deployment technology

From Workflows to Scheduling to Hybrid Cloud Resources



Processing & Automation Workflows

Any Language



Secured Data Transfers



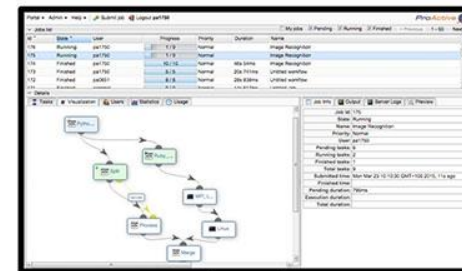
Third-part Schedulers



Market ETL, ERP...



Services Chaining & Integration



Scheduler & Orchestration

Priority & Planning Parallel Executions Error Management Multi Users



Resource Management & Monitoring

Multi Platforms



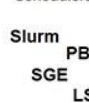
Local machine



Network Resources



Batch Schedulers



Clouds



Agenda

**I. Motivation and Requirements: flexible
analytics for big data streams**

II. Background technology: GCM/ProActive

 **III. Proposition: GCM Streaming platform** 

IV. Current work & conclusion

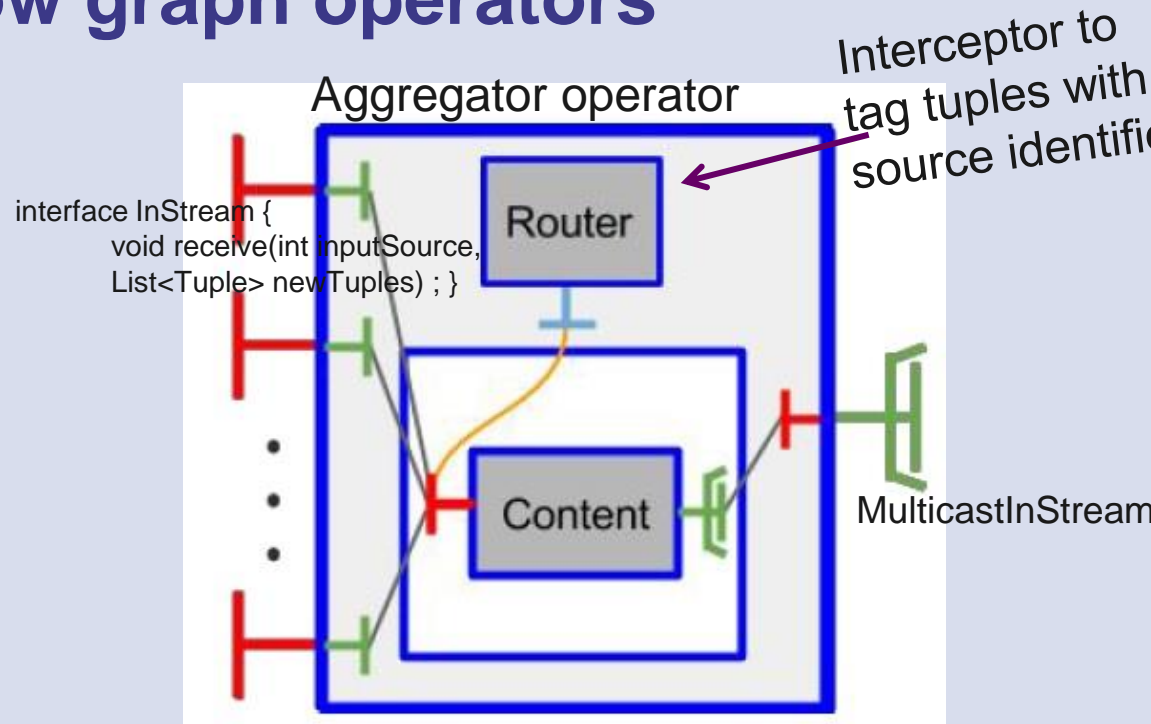
Design principles for GCM-Streaming

A platform for autonomic big data stream analytics

- Built as a GCM/ProActive application
 - Generic GCM components
 - Abstract class as content
 - Further personalized as concrete class
 - Composed together, and in a hierarchical manner
 - Whose reconfiguration can be programmed in the respective component membranes
 - Whose execution is parallel and distributed
-

Data Flow graph operators

- InTap
- OutTap
- MapReduce flavor
- Operator
- Aggregator



- All have a multicast interface as client, to propagate tuples further in the data flow graph
 - Possible configurations: broadcast, scatter/shuffle, ...
- Exposing non-functional interfaces (e.g. Attribute control)
- Specific Operator type: to run any window strategy

Programming and composition of operators

- Architecture Description File for XML-based composition

```
<!-- Twitter -->
  <component name="twitter" definition="org.inria.scale.streams.InTap">
    <content class="org.inria.scale.streams.intaps.TwitterStreaming" />
    <attributes signature="org.inria.scale.streams.configuration.TwitterStreamingConfiguration">
      <attribute name="consumerKey" value="" />
      <attribute name="consumerSecret" value="" />
      <attribute name="accessToken" value="" />
      <attribute name="accessTokenSecret" value="" />
      <attribute name="terms" value="tsipras,bieber,obama,messi,tevez" />
    </attributes>
  </component>
<!-- Input window -->
  <component name="input-window" definition="org.inria.scale.streams.Window">
    <attributes signature="org.inria.scale.streams.configuration.WindowConfiguration">
      <attribute name="windowConfiguration" value="( 'type': 'tumbling', 'tumblingType': 'time', 'milliseconds': 10000 )" />
    </attributes>
  </component>
<!-- Getter -->
  <component name="getter" definition="org.inria.scale.streams.Operator">
    <content class="org.inria.scale.streams.operators.Getter" />
    <attributes signature="org.inria.scale.streams.configuration.GetterConfiguration">
      <attribute name="tupleComponent" value="0" />
    </attributes>
  </component>
.....
<!-- Operator bindings -->
  <binding client="twitter.out" server="input-window.in" />
  <binding client="input-window.out" server="getter.in" />
  <binding client="getter.out" server="normalizer.in" />
.....
```

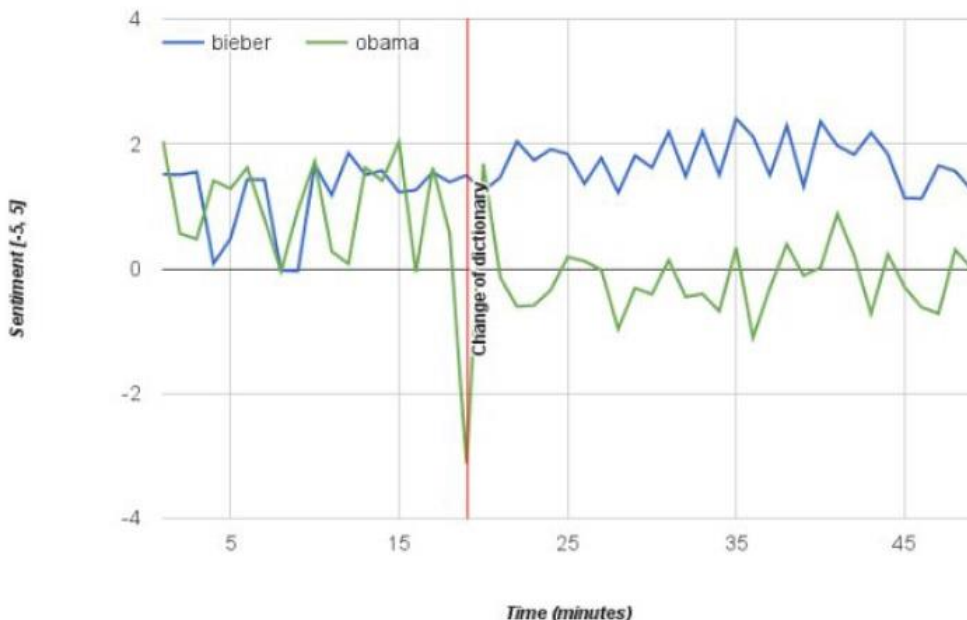
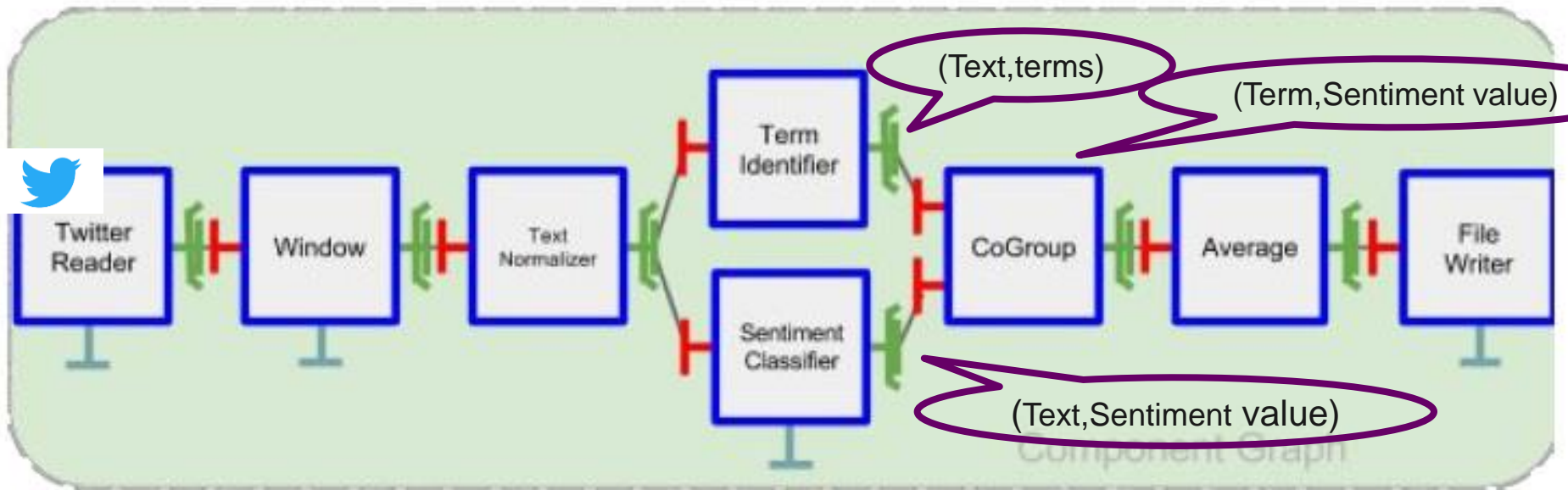
```
public class Getter extends BaseOperator implements
GetterConfiguration {
```

```
    private int tupleComponent;
```

```
    @Override
```

```
    protected List<? extends Tuple> processTuples(
        final List<Tuple> tuplesToProcess) {
        return .....
```

Sentiment analysis GCM-streaming application



- Rank the sentiment (negative -5 to positive +5) in Spanish tweets pertaining to popular terms,
 - E.g.: Bieber, Obama
- At minute 20, dynamic adaptation of the dictionary used => English
- Obama, a personality which seems to be more appreciated by Spanish speakers

Future work around GCM-Streaming

GCM-Streaming: Public code: <https://github.com/moliva/gcm-streaming>

- Better benefit from Multi-active objects for parallel tuples processing within an operator, besides operator fission
 - extend the multi active object requests scheduler
 - Advanced request parameters & operator state based compatibility rules
- DSLs for easing generic components implementation programming

Towards predictive analytics

- Deeply explore functional scenarios from e.g. **situation-aware computing**
 - Expression of autonomic GCM-streaming reconfiguration policies
 - triggered by functional concerns
 - Safe protocols to transfer state and awaiting tuples

Need of manpower to pursue this work, in the Scale Team

An open PhD funded position !!!!!!!

<https://team.inria.fr/scale/files/2011/07/GCMStreamingPhD.pdf>

from Fall 2016