AMEBA Ideas. Canals and Smart Spaces.

Natalia Díaz Rodríguez, Andreas Dahlin, Johan Ersfolk
ndiaz, andreas.dahlin, johan.ersfolk@abo.fi
Åbo Akademi University, Turku, Finland

Turku, 30.1.2011.

1 Introduction

This paper explores work possibilities for the AMEBA project by combining ideas from Canals and Smart Spaces end-user programming.

2 Programming data-stream applications with Canals: Embedding Semantics and Event-based programming into Data-flow streams.

The AMEBA project demonstrator consists of an application whose aim is having a video stream input in which semantic annotations are made in runtime. Annotations of the video can include, from faces of familiar people appearing in the images to GPS location data associated to the moment when the recording was done. A rule editor interface allows the end-user to query for metadata such as, e.g., objects, people, or faces appearing in the video in certain location and time. Specifically, the application has two phases:

1. Data-flow based processing: Decoding of the video and simultaneous registration of metadata, saving semantic annotations into the Smart Space store.
2. Event-based processing: Allow search and creation of rules through the Smart Space end user interface. For example:
   -"IF there is images of me from yesterday, THEN show them to me."
   -"If Juuso appears in videos in folder X, THEN show scenes with start and end time".
   -"IF Åbo Akademi professors visited the ICT Showroom corridors, THEN show the times when they appear visiting the stands".

The diagram in figure 1 shows how both data-stream and event-stream architectures can be merged to serve such application scenario in a Canals language network. Some considerations are:

- The application is modeled as a Canals Network with 2 inputs and 1 output. Since Canals only allows 1 bitstreams input/output, inputs need to be intercalated in time.
- Timestamp must be obtained from images and GPS coordinates. Inputs with Person+Time or Location+Time need to be intercalated in time as well.
- `MultipleAnnotate()` function: Called when an image and a GPS coordinates are found under the same timestamp interval. This function takes into account a threshold value for determining under what interval of time an image can be associated to a person appearing in it. Threshold time can be optionally given by the user (otherwise some default value, e.g., 1h). A multiple annotation can include several Triple annotations, e.g.: `Person-appearsIn-Location`, `Person-appearsIn-Time`, `Person-appearsIn-Image`.
- `SimpleAnnotate()` function: Called when an image is not found at the same timestamp as the GPS coordinate, i.e., only an image or a GPS coordinate has been found

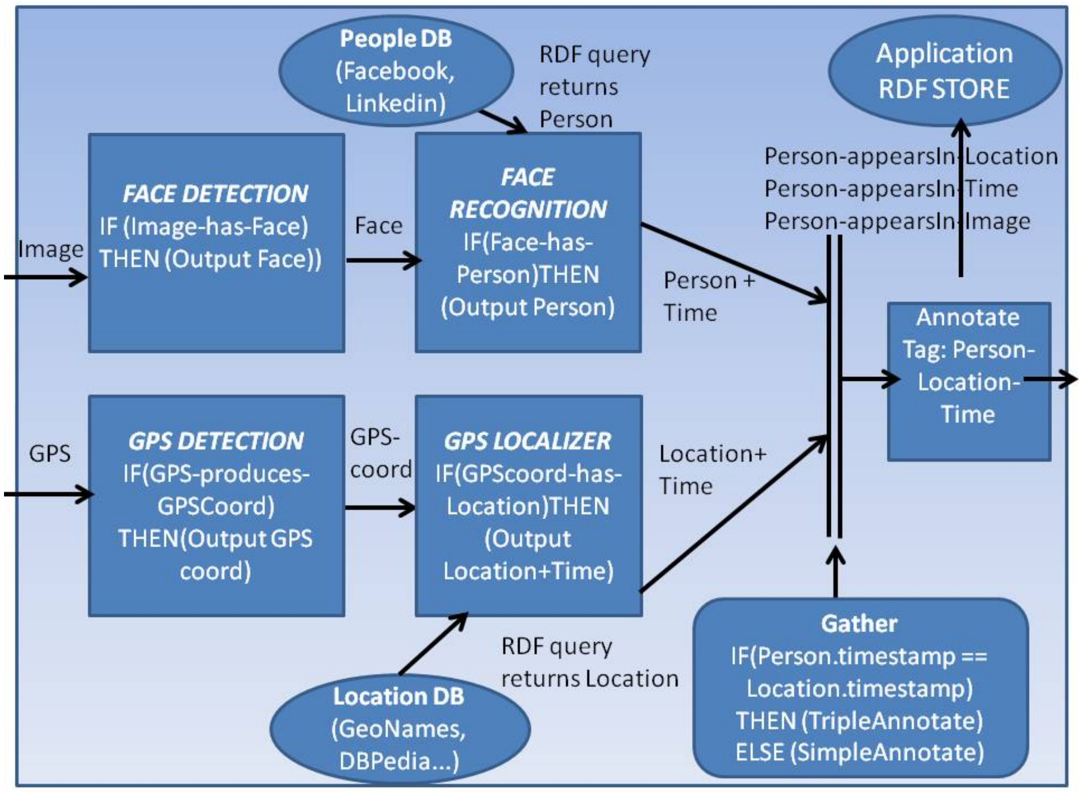
People DB (Facebook, Linkedin)
RDF query returns Person
Application RDF STORE
FACE DETECTION IF (Image-has-Face) THEN (Output Face))
Image
Face
FACE RECOGNITION IF(Face-has-Person)THEN (Output Person)
Person-appearsIn-Location
Person-appearsIn-Time
Person-appearsIn-Image
Person + Time
Annotate Tag: Person-Location-Time
GPS
GPS DETECTION IF(GPS-produces-GPSCoord) THEN(Output GPS coord)
GPS-coord
GPS LOCALIZER IF(GPScoord-has-Location)THEN (Output Location+Time)
Location+ Time
Gather IF(Person.timestamp == Location.timestamp) THEN (TripleAnnotate) ELSE (SimpleAnnotate)
RDF query returns Location
Location DB (GeoNames, DBPedia...)

Figure 1: Data Stream and Event-based face recognition on a video stream.

under the same threshold interval of time. A simple Annotation is done when a unique Triple is created: `Location-appearsIn-Time, Person-appearsIn-Picture`, or `Person-appearsIn-Time`.

- Some external modules/kernels (as decomposed "black boxes") to be connected within a Canals network are:
  - The Smart Space Kernel (including the RDF repository, the rules and the rule engine). The application's general RDF store will be available outside the application as a web service for the end-user to edit, create, execute and save his own rules.
  - The image detection and image recognition algorithms. The open source TLD (or "Predator") and FaceTLD algorithm, using unsupervised learning, serve this purpose. Predator is implemented in Matlab and C [3, 6]. Another option is using Kinect's KAET [1](Kinect Annotation and Evaluation Tool), which allows annotation of images (containing one or more persons) and simplifies training and evaluation of machine learning algorithms, with annotations for 2D and 3D pose estimation. Samsung's OS Bada also provides face detection and recognition algorithms. We will study FaceTLD for platform and camera independency reasons.
  - When external databases are queried, an input and output into the same kernel should be modeled (since kernels only have one input and one output). For example, when the kernel receives GPS coordinates and should return Position to the same computational node who asked for it. One way is serializing SPARQL as a datatype to stream (since Rules can also be serialized as SPARQL queries [2]). Implementation through SPARQL Queries and SPIN Rules
  - Communication mechanism between Canals-RDF store: Rules can be mapped into SPARQL queries through SPIN rules (and SPIN RDF converter (http://spinservices.org/spinrdfconverter REST API)). Options: A. To call REST API from Canals or B. To have only SPARQL dataflow interchange in between Canals different kernels and the RDF stores.

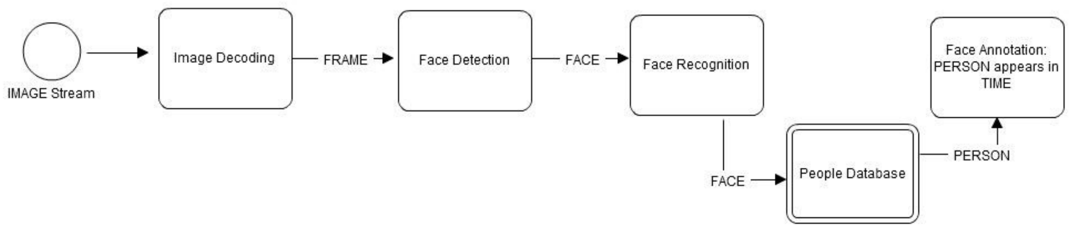

Figure 2: Application Stream Data Flow in BPMN (Business Process Model Notation).

Synergy between Stream processing and Event-driven processing		
Feature	**Data-Stream**	**Event-driven processing**
Main unit of computation	Kernel of Computation	Rule
Main unit of data storage	Kernel of Computation	Rule
Triggering mechanism	Data availability + conditions	Conditions
Possibility of parallelization (of main unit of computation)	Yes, if data is independent of each other	Yes, if rules are domain-independent
Synchronization mechanism	Synchronous? Architecture dependent synchronization. Via the scheduler and guards	Asynchronous. Via Publish/-Subscribe notifications
Control flow	Through Kernel-Channel network architecture and scheduler	Via Publish/Subscribe notifications
Execution	Dispatcher	Rule engine, external control flow

Table 1: Synergy between Stream processing and Event-driven processing

2.1 Motivation: Synergy between Data-stream processing and Event-driven architectures

The motivation for joining three kind of paradigms is to take advantage of the different potential provided by different models of computation. The two main models of computation, stream processing and event processing, can be represented simply as figures 2 and 3 show. Although stream processing and event-driven processing are characteristically focused on different purposes [4], we can appreciate that both data flows are not so different, for our use case, and they can be integrated together.

Our aim is to study the synergy and advantages of using data-stream processing together with an event-driven architecture in which we can add semantics. The main issues to study are: What is the proportion of integration needed, in what form? How do event based programming and dataflow can properly communicate to complement each other's functionality? We want to analyze if any of this models of computation can lead (or be subsumed into) the other.

To make the analogy more specific, the table 1 reflects the synergy between the main abstractions of both paradigms. To make the stream processing comparison more specific, we use terminology from the stream data-flow Canals language. Regarding event based paradigms, we focus our attention on ECA (Event Condition Action) with publish/subscribe architectures.

2.2 The Canals language

Canals [5] is based on the concept of nodes (kernels and networks) and links (channels), which connect nodes together. Computations are performed in the nodes and the links

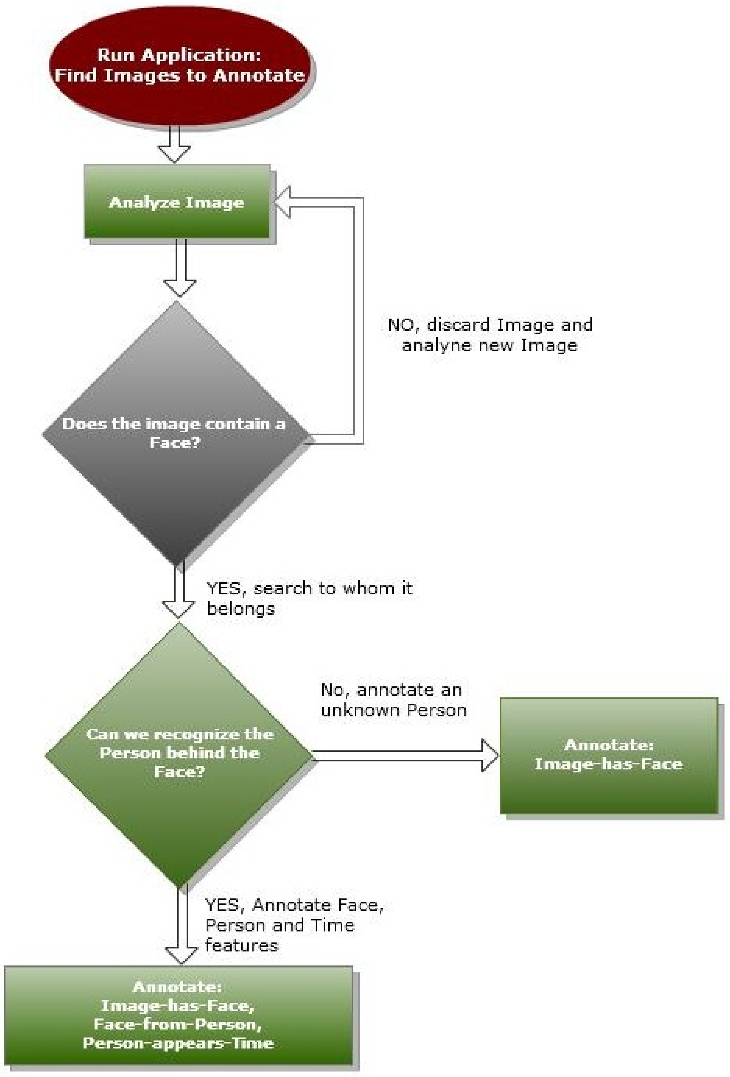
Run Application:
Find Images to Annotate
Analyze Image
NO, discard Image and analyne new Image
Does the image contain a Face?
YES, search to whom it belongs
Can we recognize the Person behind the Face?
No, annotate an unknown Person
Annotate: Image-has-Face
YES, Annotate Face, Person and Time features
Annotate: Image-has-Face, Face-from-Person, Person-appears-Time

Figure 3: Application Event-based processing in a flow chart.

are representing intermediate buffers in which output data from nodes is stored before the data is consumed by the next node. Since Canals is a language independent platform, when a Canals program is mapped to an architecture, the platform independent links are mapped to real connectors on the architecture. Connectors are implemented in the Hardware Abstraction Layer (HAL). HAL consists of stateless low-level functions that triggers underlying hardware mechanisms specific for the underlying platform and the functions used by the dispatchers execute kernels, handle synchronization of kernels/processors and triggers memory transfers. A Canals program without a mapping does not depend on specific hardware synchronization concepts but, instead, only describes the intended behavior of a program in a platform independent manner. Summarizing, the main features of Canals are: the ability to describe scheduling algorithms within the language itself, the implicit deserialization of data at network inputs, and an abstraction layer for synchronization between scheduler and computations.

2.3 Advantages from integrating Stream and Event based programming approaches

Some of the benefits we can obtain from merging these two paradigms are:

- Event-driven processing requires usually some programmatic background application for driving the general control flow. Stream processing can fill that gap to complement a rule engine by not only providing a dataflow stream but, at the same time, to control the underlying flow of the application's algorithm.

- Since Publish/Subscribe architectures work as a continuous polling, such as the observer/listener pattern, a dataflow stream needs to be active (i.e. reading and writing) during the whole run time of the application. Therefore, publish/subscribe paradigm can be easily integrated into stream processing applications in form of control-data flow.

- Since Canals describes the intended behavior of a program in a platform independent manner, all elements exist in parallel and are capable of performing computations concurrently from a resource point of view. Due to the fact that only data dependencies restrict the parallelism [1], multicore architectures can benefit of Canals specification language to parallelize independent tasks hierarchies. Furthermore, this hierarchy can be applied to multicore and 3D chips architecture so that rules from different domain can be organized in different hierarchy levels that can be parallelized. Rules can be related,for example, to metadata processing, data processing or other internal processing such as synchronization, data flow or control flow. In this way, hierarchically independent data flows, with different purpose, can be run concurrently.

- Time is a dimension which is not easily taken into consideration within semantic web programming approaches. By combining semantic event-based programming with stream-programming, time awareness can be added, as another data flow, in a easier and more synchronized way.

2.4 Application scenarios for Stream and Event based programming

The synergy of both mentioned architectures has been studied previously [4]. Our study is basically targeted to 3D data storage systems with a distributed agent-based data management. The main focus goes into: 1. Highly parallel integrated SSD based data storage architectures, 2. Using agent based energy efficient and dependable data management and retrieval methods, and 3. Domain-specific languages for quality-of-service (QoS) aware distributed applications.

To concretize more, in relation with our application demonstrator, we propose different QoS features to model:

- *Energy Versus Performance awareness.* Allow a variability of algorithms for different performance/energy requirements. For example, we can have different algorithms for face recognition, some computationally more expensive (and better performance) than others. Depending on real time context execution requirements, which can be set by the user or by the system, an algorithm or another can be chosen.

- *Answer Completeness.* The query of data or invocation of a service can be classified into two different categories:
  - Require *Complete* answer: When the user or the system requires a complete answer, the algorithm needs to be executed completely for all data available in the storage.
  - Require *Incomplete* answer: If the answer is required to be incomplete (as usually will be, by default), partly and/or enough results of the query, or execution, will be given as output. Normally, restriction can be due to different factors, such as memory, resources or energy limitations. Energy costs can be balanced by not showing all results (or run process over all data), but producing practically useful results. Another way to lower costs, e.g., in low capacity devices, is by powering down unnecessary computation nodes, memories or processors. Incompleteness gives the advantage of energy optimization.

- *Data Semantics.* Data-flow should acquire semantics for allowing context metadata information retrieval. Semantic queries must be optimized by an event processing

[1] In Canals, the completely concurrent behavior can be restricted in the compilation process by supplying a mapping and an architecture description to the compiler

system that integrates semantic metadata. Different ontologies relating multimedia stream data can be arranged in different nodes, allowing intelligent accesses and cache management.

- *Response Time and Latency.* The system must be resource aware in order to allocate, access and process resources in an optimized way, as well as to reduce power loss.

3 Contribution obtained by inserting semantic Smart Spaces programmability into Canals

Questions open to answers are: What is the main purpose of this experiment? What is our contribution?

- Ability to process semantic data in multicore programming?
- A stream language with semantic support?
- A tool for online image recognition through semantic annotations in a video stream. This can be made with independence of image recognition algorithm and independence of database, since Canals is language independent.
- A concrete application: by combining the execution of Canals with simultaneous semantic annotation allows for a metadata and information retrieval system with possibilities of integration with third party applications within a Smart Space.
- Could we visualize every core in a multicore architecture as a Canals kernel to parallelize the annotation?
- The semantic-stream architecture can be generalized and used in ubiquitous computing for sensor networks, continuous sensor observations, etc.
- Another use case for the video stream can be in the bioimaging domain, taking continuous sets of image streams from the microscope for an automatic selection and/or tagging of relevant images.
- What is the advantage to model the video application in data flow language as opposed to standard C++ or Java?
- Specific sounds could also be searched. How to represent sound in a semantic-stream way?
- Could we save in memory access when different cores with different memories execute parallel processing (decoding, annotation and search) of frames in Canals? Can Canals be reprogrammed in run-time regarding platform independence (i.e., number of cores/kernels)?

The inconvenient is that, if we do not want to modify Canals syntax, the face detection and the metadata annotation (calling the Smart Space kernel) should be done in a single kernel, which means that this process will not be able to be parallelized. For the moment, since we also would like to keep the image recognition system as a "black box" kernel without having to translate it completely to Canals, only the frame decoding phase would be able to be parallelized.

References

[1] KAET. kinect annotation and evaluation tool. [online]: http://www.multimedia-computing.org/wiki/kaet.

[2] SPIN [online]: http://www.spinrdf.org/.

[3] TLD or predator algorithm. [online]: http://info.ee.surrey.ac.uk/personal/z.kalal/tld.html.

[4] S. Chakravarthy. Events and Streams : Harnessing and Unleashing Their Synergy ! *Event (London)*, pages 1–12, 2008.

[5] A. Dahlin, J. Ersfolk, G. Yang, H. Habli, and J. Lilius. The Canals Language and its Compiler. *Language*, 2009.

[6] Z. Kalal, K. Mikolajczyk, and J. Matas. FACE-TLD : TRACKING-LEARNING-DETECTION APPLIED TO FACES Centre for Vision, Speech and Signal Processing, University of Surrey, UK Center for Machine Perception, Czech Technical University, Czech Republic. (i).