ColisTrack: Testbed for a Pervasive Environment Management System

Yann Gripay[1,2], Frédérique Laforest[1,4], Francois Lesueur[1,2], Nicolas Lumineau[1,3], Jean-Marc Petit[1,2], Vasile-Marian Scuturici[1,2], Samir Sebahi[1,2], Sabina Surdu[1,2,5]

[1]*Université de Lyon, CNRS*
[2]*INSA-Lyon, LIRIS, UMR5205, F-69621, France*
[3]*Université Lyon 1, LIRIS, UMR5205, F-69622, France*
[4]*Telecom Saint-Etienne, UJM, LT2C, F-42000, France*
[5]*Babes-Bolyai University, Cluj-Napoca, 400084, Romania*
[1]firstname.lastname@liris.cnrs.fr

ABSTRACT

One of the leading challenges for pervasive computing is to ease the application development to smoothly handle the surrounding environment. We consider the case where the environment produces heterogeneous and continuous data, *e.g.* temperature readings, car positions... We have defined a scenario for containers transportation tracking in a medical context involving the transportation of fragile biological matter in sensor-enhanced containers. This scenario has been simulated as a testbed and offers a very nice setting to measure the agility of data-centric application development.

On top of this scenario, we have built a pervasive application using a Pervasive Environment Management System called SoCQ (Service oriented Continuous Queries). SoCQ provides a data-oriented perspective of the pervasive environment, mixing classical data, streams and functionalities. For the demo, our objective is twofold: first, from the application developer point of view, she has access to the underlying SoCQ-schema and she may pose her own SQL-like queries to the simulated environment. Second, from the end-user point of view, she may quite easily interact with the environment either through a general dynamic visualization with Google Maps of hospitals, cars moving along roads and medical containers waiting or being transported, or by getting SMS notifications on her own phone of results of predefined queries.

Categories and Subject Descriptors

H.2.4 [**Information Systems**]: Database Management Systems—*Query processing*

Keywords

Data Streams, Continuous Queries

1. INTRODUCTION

One of the leading challenges for pervasive computing is to ease the application development to smoothly handle the surrounding environment. The notion of *agility* is convenient and now widely used to capture the easiness of application development. Indeed, agility may have a variety of dimensions, from model agility to operational agility and programming agility [6]. In this setting, we are interested in data-centric applications involving data from a wide variety of data sources: from classical databases to streaming data and distributed data services available in some sensor networks or the Internet.

While discussing on the interest in such environments, referred to as pervasive environments [7], is quite interesting and easy to do, the first bottleneck we need to address is to define, as rigorously as possible, a scenario involving all the necessary ingredients: data, streams and services in a distributed environment. For this demo paper, we have defined such a scenario, called *ColisTrack*, whose aim is to track *containers transportation in a medical context*. It encompasses the transportation of fragile biological matter in sensor-enhanced containers.

This scenario has been simulated as a testbed with a wide range of options (*e.g.*, number of containers, speed of cars, number of cities) and it offers a nice setting to measure both the operational and application agility. Moreover, even if ColisTrack shares some common objectives with classical benchmarks, ColisTrack is oriented towards agility dimensions and is just a testbed, not a benchmark like LinearRoad [1] or TPC variants, *i.e.*, we do not provide any queries and measures like LinearRoad does. Indeed, our testbed provides a framework to simulate data, streams and services with convenient tools to write queries and to monitor a running instance of the simulated data. The detailed description of this testbed is out of the scope of this paper though.

On top of this testbed, a data-centric pervasive application has been designed using the SoCQ (Service oriented Continuous Queries) data model [4]. SoCQ allows to combine data, streams and distributed services in a unified model. It provides a declarative query language "à la SQL" to homogeneously handle data, streams and services. To stay as close as possible with database notions, the notion of "relational pervasive environment" composed of several eXtended Dynamic Relations (XD-Relations for short) has been defined to model such a pervasive environment.
More importantly, a PEMS (Pervasive Environment Management System, including the SoCQ Query Engine) has been developed and will be used for the demonstration. It will be the main contribution of the demonstration on which our objective is twofold: first,

EDBT 2012, March 26–30, 2012, Berlin, Germany.

from the application developer point of view, she has access to the underlying SoCQ-schema (mixing classical data, streams and services) and she may pose her own SQL-like queries to the simulated environment. Second, from the end-user point of view, she may quite easily interact with the environment either through a general dynamic visualization with Google Maps of hospitals, cars moving along roads or medical containers waiting or being transported, or by getting SMS notifications on her own phone of results of predefined queries.

The interesting point of the demonstration is a real Google Maps view of Lyon, Turin, Zurich and Geneva with animated cars transporting medical containers and a real-time interaction with the person who will attend the demo: she will receive an SMS on her phone notifying temperature problems on the containers she wants to track (see Fig. 3).

To the best of our knowledge, the scenario, the simulation of the scenario and its convenient visualization is a contribution per se for data-centric pervasive applications. Moreover, it points out the feasibility of the SoCQ data model introduced in [4] as a way to simplify the application development over distributed data sources, at the price of database design. Last but not least, the architecture has been thought as open as possible (see Fig. 1) to offer a platform to evaluate other systems, from classical DBMSs with ad-hoc programming to research prototypes (*e.g.*, [2, 5]).

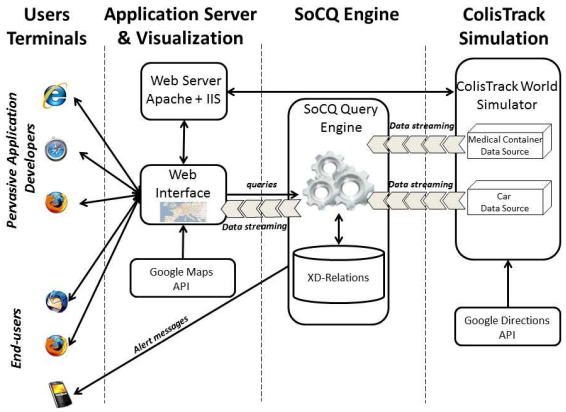

Figure 1: ColisTrack deployment

2. COLISTRACK SCENARIO AND TESTBED

2.1 Motivating scenario: Medical Containers Tracking

Nowadays, more and more quality norms are specified in industrial tasks, in order to ensure a secure management of these tasks and avoid risks of failure. Respecting those norms is essential in specific domains like medical and pharmaceutical environments where people's lives are concerned. For instance, when a blood sample is taken at home or in a medical laboratory, the reliability of the results depends on the quality of the transportation. Our motivating scenario is about the transportation of fragile biological matter in sensor-enhanced containers. Quality criteria for biological matter transportation are defined by legislation. Blood, platelets, progenitor cells or organs are living matter which has to be quickly handled by technical biologists before some given deadline; they cannot tolerate too high or too low temperatures and cannot be shaken. As the

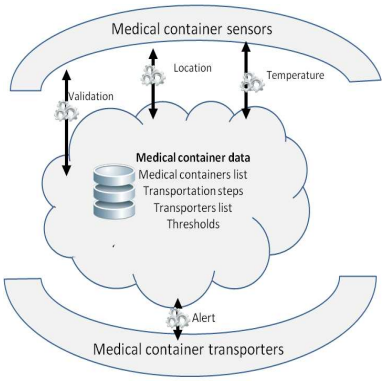

Figure 2: Medical container scenario

transported matter is alive, time out has to be considered to ensure its freshness.

During the container transportation, temperature, acceleration and time must be observed. The corresponding sensors are embedded in the container: a temperature sensor to verify temperature variations, an accelerometer to detect high acceleration or deceleration, a timer to control the deadline beyond which the transportation is unnecessary and a GPS to know the container position at anytime. Moreover, each container contains memory to store event logs.

The transportation is divided into succeeding steps, each step being ensured by a different transporter. At each step, the corresponding transporter is responsible for the container and has to monitor quality criteria. Sensors on the container can emit alerts that are forwarded to the responsible transporter.

For each step, a supervisor determines thresholds for the different quality criteria the container must meet. When a threshold is exceeded, the container sends a text message via SMS to the current transporter. A critical threshold specifies the bound that a criterion must not exceed (*e.g.*, some cells must not be exposed to more than 37°C). Then, a first transporter takes the responsibility for the container until taking over to the next transporter. Each validation step is done when a transporter takes the responsibility of the container. To validate the state of a container, the event log is read to detect any failure.

In this scenario (see Fig. 2), only little information is static and can be stored in classical databases: the medical containers descriptions, the transportation steps and transporters, the different thresholds. All other data in this environment are dynamically produced by distributed services and accessed through method invocations (*e.g.*, get current location) or stream subscriptions (*e.g.*, temperature notifications). Moreover, services can provide additional functionality like sending some messages (*e.g.*, by SMS) when an alert is triggered.

2.2 Testbed and technology

The testbed implements the ColisTrack scenario as follows. On the server side running on Windows 2008 server, a simulation engine has been developed as a C# application. The different options to set up ColisTrack can be specified in a XML file, like the number of cars, the places they visit, the generation of medical containers, etc. The engine uses the GoogleMaps directions API REST Web

Service[1], in order to compute real routes of cars.

The user interface is implemented as a Web application that allows the visualization of the simulated ColisTrack environment. It runs on an Apache Web server and the server side is developed in PHP. On the client side, our web user interface is based on the GoogleMaps API to visualize the simulated ColisTrack on a map, and uses Ajax XML HTTP Request in order to load the simulated ColisTrack state from the server side. Several remote clients can connect simultaneously to the same simulated environment, by simply using their Web browser.

By using a REST/HTTP-based protocol, our framework enables the integration of data services independently of the operating system and of the used programming language. HTTP has been designed to work in a pull manner: the client sends a request to the server, waits for the response, processes the response and eventually closes the connection. In our approach, data services can also work in a push manner: the consumer connects to a data service and waits indefinitely to receive produced data. We simulate this kind of streaming using a permanently open HTTP connection between the consumer and the data service. The data service sends every newly produced tuple to the consumer and keeps the connection open.

Each data service is identified by an URL and accepts a set of operations via HTTP. The main operations are the following:
- Enumerate the resources published by the data service (methods, streams),
- Get the schema (input and output attributes) of a resource,
- Invoke a method or subscribe to a stream to retrieve data.

3. APPLYING SOCQ ON THE TESTBED

The PEMS implementing SoCQ is based on a distributed data access model which uses REST/HTTP data services and is therefore fully compatible with the testbed.

In the rest of this section, a quick overview of SoCQ is given, the interested reader should refer to [3, 4] for details. Then, a SoCQ schema of ColisTrack is proposed and the easiness of application development is shown through a couple of SoCQ queries.

3.1 SoCQ overview

With SoCQ, the basic notion is called XD-Relation, which is basically a relation with some new features. (1) Its attributes can be either real or *virtual*. Virtual attributes represent input and output parameters of *service resources* that may receive some values through query operators. (2) Its schema can be further associated with *binding patterns* indicating which service resources are involved (with which input/output attributes). *Service identifiers* (*i.e.*, service URLs) are handled as data values of some predefined attributes. XD-Relations may be either finite (*i.e.*, like a standard relation) or infinite (*i.e.*, a data stream). Currently, binding patterns represent either invocations of a method resource or subscriptions to an output stream resource.

Standard relational operators have been redefined over XD-Relations and new operators dedicated to virtual attributes and binding patterns have been introduced. Among them, the *service discovery operator* can build XD-Relations that represent a set of available services providing some required resources. For example, a XD-Relation SENSORS could be the result of such an operator and be continuously updated when new temperature sensor services become available and when previously discovered services become unavailable.

[1]http://code.google.com/apis/maps/

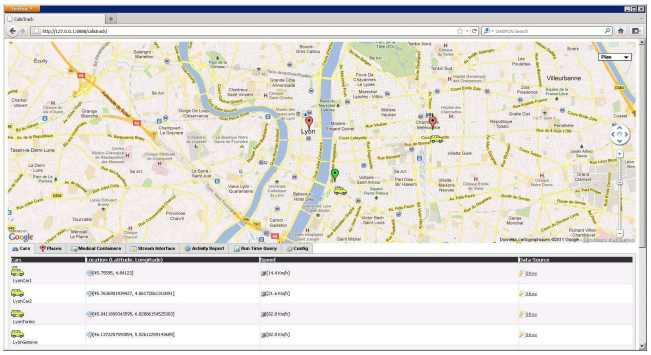

Figure 3: ColisTrack Web Interface

3.2 SoCQ schema for ColisTrack

To perform our scenario, a set of XD-Relations has been defined and implemented within our PEMS. A description of the SoCQ schema is given in Fig. 4. Four XD-relations have been designed:
1) Car: cars are identified by a carID and their associated data services provide a GPS location stream (see Fig. 4.(a)).
2) $MedicalContainer$: medical containers are also identified by a mcID, and their associated data services provide temperature as a stream, and location, time to live and temperature again as methods (see Fig. 4.(b)).
3) $SupervisorMobile$: supervisors are associated with a messenger service in order to receive alert messages on their phone (see Fig. 4.(c)).
4) $Supervise$: it is a classical relation and associates some medical containers with supervisors (see Fig. 4.(d)).

The interplay between virtual attributes, services and binding patterns appears clearly for the first three statements. In order to automatically populate XD-relations Car, $MedicalContainer$ and $SupervisorMobile$ from a running testbed, we rely on the service discovery operator [4]. When new services are available in the simulation, tuples with the service URLs are inserted into the corresponding XD-Relations. Note that the database design principles to be used to devise such a schema are out of the scope of this paper.

3.3 SoCQ queries

Devising an application on top of ColisTrack is now as easy as writing a SQL-like query. Three examples are given in Fig. 5.

For instance, to track cars positions, we use a view called $carSupervision$ which is actually a stream defined as a simple SP query involving the binding pattern $locationNotification$ (see query (a), Fig. 5). Detecting critical temperatures during the transportation of medical containers is also easy with a SPJ query: a view is created which provides a stream of tuples whenever some conditions are met, among them when the threshold is exceeded, and sends alert messages to supervisors (see query (b), Fig. 5). Classical one-shot queries can also be defined, for instance to get the current location of a medical container (see query (c), Fig. 5).

```
CREATE RELATION Car (
 carID STRING PRIMARY KEY,
 carService SERVICE,
 latitude STRING VIRTUAL,
 longitude STRING VIRTUAL,
 locDate DATE VIRTUAL
)
USING BINDING PATTERNS (
 locationNotification[carService]() :
 (latitude, longitude, locdate)TREAMING
);
```

(a)

```
CREATE RELATION MedicalContainer (
 mcID STRING PRIMARY KEY,
 mcService SERVICE,
 temperatureDate DATE VIRTUAL,
 temperatureValue REAL VIRTUAL,
 latitude STRING VIRTUAL,
 longitude STRING VIRTUAL,
 locDate DATE VIRTUAL,
 timeout REAL VIRTUAL
) USING BINDING PATTERNS (
 temperatureNotification[mcService] () :
 (temperatureDate, temperatureValue) STREAMING,
 getTemperature[mcService] () :
 (temperatureDate, temperatureValue),
 getLocation[mcService]() :
 (latitude, longitude, locdate),
 getTimeout[mcService]() :
 (timeout, timedate); );
```

(b)

```
CREATE RELATION SupervisorMobile (
 mobileID REAL PRIMARY KEY,
 phone STRING, alertService SERVICE,
 alertDate DATE VIRTUAL,
 alertMessage STRING VIRTUAL,
 alertSent BOOLEAN VIRTUAL
) USING BINDING PATTERNS (
sendSMS [alertService](phone, alertDate, alertMessage):
 (alertSent) );
```

(c)

```
 CREATE RELATION Supervise (
 mobileID STRING,
 mcID STRING,
 temperatureThreshold REAL,
 fromLatitude STRING,
 fromLongitude STRING,
 fromDate DATE,
 toLatitude STRING,
 toLongitude STRING,
 toDate DATE,
 PRIMARY KEY (mobileID, mcID)
 );
```

(d)

Figure 4: SoCQ schema for ColisTrack

```
CREATE VIEW STREAM carSupervision(
 carID REAL,
 locDate DATE,
 locLatitude STRING,
 locLongitude STRING
) AS
 SELECT c.carID, c.locDate, c.latitude, c.longitude
 STREAMING UPON insertion
 FROM Car c
 USING c.locationNotification [1];
```

(a)

```
 SELECT latitude, longitude, locdate
 FROM MedicalContainer
 WHERE mcID=12345 USING getLocation;
```

(c)

```
CREATE VIEW STREAM temperatureSupervision(
 mcID REAL,
 temperatureDate DATE,
 temperatureValue REAL,
 temperatureThreshold REAL,
 temperatureSent BOOLEAN
) AS
SELECT mc.mcID, mc.temperatureDate, mc.temperatureValue,
 s.temperatureThreshold, mob.alertSent
STREAMING UPON insertion
FROM MedicalContainer mc, Supervise s,
SupervisorMobile mob
WITH mob.alertDate := mc.temperatureDate ,
mob.alertMessage := concat("Temperature error : ",
 mc.temperatureValue)
WHERE mc.mcID = s.mcID
 AND mob.mobileID = s.mobileID
 AND s.temperatureThreshold < mc.temperatureValue
USING mc.temperatureNotification [1], mob.sendSMS ;
```

(b)

Figure 5: SoCQ queries and views

4. DEMONSTRATION

A running instance of the SoCQ engine is connected to the simulated ColisTrack as depicted in Fig. 1. A Web interface displays the map of Western Europe and the main actors positions on this map (see Fig. 3). Cars are moving on the map on real routes in real traffic conditions. Medical containers are displayed in hospitals (waiting to be transported) or in cars (during the transportation). In order to interact with ColisTrack via SoCQ, a particular Web page implements an interface to SoCQ, enabling the execution of continuous queries. Here, developers can write continuous queries against SoCQ, mixing heterogeneous data sources. This interface also enables application developers to see the underlying SoCQ schema.

The application is multi-user: the same simulated environment is visible to all users, with different view settings (selected region to be displayed, zoom parameters, etc.). The URL of the demonstration will be available during the demonstration session, and free access to the ColisTrack Web interface will be granted to all participants.

After a brief presentation of the SoCQ query engine and the scenario, the audience will be invited to interact with the demo using the SoCQ Web interface and run their own queries against the testbed.

5. REFERENCES

[1] A. Arasu, M. Cherniack, E. F. Galvez, D. Maier, A. Maskey, E. Ryvkina, M. Stonebraker, and R. Tibbetts. Linear road: A stream data management benchmark. In *VLDB*, pages 480–491, 2004.

[2] V. Cuevas-Vicenttin, G. Vargas-Solar, C. Collet, N. Ibrahim, and C. Bobineau. Coordinating services for accessing and processing data in dynamic environments. In *OTM Conferences (1)'10*, pages 309–325, 2010.

[3] Y. Gripay. *A Declarative Approach for Pervasive Environments: Model and Implementation*. Phd thesis, INSA de Lyon, 2009.

[4] Y. Gripay, F. Laforest, and J.-M. Petit. A Simple (yet Powerful) Algebra for Pervasive Environments. In *EDBT 2010*, 2010.

[5] O. Jurca, S. Michel, A. Herrmann, and K. Aberer. Continuous query evaluation over distributed sensor networks. In *ICDE*, pages 912–923, 2010.

[6] M. Rys. Scalable SQL. *Commun. ACM*, 54(6):48–53, 2011.

[7] M. Weiser. The Computer for the 21st Century. *Scientific American*, 265(3):94–104, September 1991.