

Towards an ambient data mediation system

Kim Tâm Huynh
PRiSM Laboratory
University of Versailles
Saint-Quentin-en-Yvelines
Versailles, France
kth@prism.uvsq.fr

Béatrice Finance
PRiSM Laboratory
University of Versailles
Saint-Quentin-en-Yvelines
Versailles, France
beatrice@prism.uvsq.fr

Mokrane Bouzeghoub
PRiSM Laboratory
University of Versailles
Saint-Quentin-en-Yvelines
Versailles, France
mok@prism.uvsq.fr

ABSTRACT

In this paper, we address the problem of integrating many heterogeneous and autonomous tiny data sources, available in an ambient environment (AmI). Our goal is to facilitate the development of context-aware and personalized embedded applications on mobile devices. The originality of the approach is the new ambient mediation architecture which provides declarative and dynamic services, based on rules/triggers. These services provide facilities to develop and deploy ambient applications over devices such as smartphones. This paper reports also on our first experimental prototype, combining Arduino+Android.

1. INTRODUCTION

Over the last 20 years there have been some significant progresses on the miniaturization of hardware components and wireless networks. The number and capabilities of mobile devices, wireless sensors and sensor networks open new research fields and applications. Terms such as the “Web of sensors”, the “Internet of Things” and “Ambient Intelligence (AmI)” emphasize the trend towards a tighter connection between the cyberspace and the physical world.

Today, we are witnessing an unprecedented explosion of mobile data volumes (i.e. ambient data). According to a study from ABI Research [1], in 2014 the volume of mobile data sent and received every month by users around the world will exceed by a significant amount the total data traffic for all of 2008. In 2011, 1.08 billion of mobile phone users have a Smartphone and in the near future they will be surrounded by many sensors/actuators.

In his survey, Sadri [18] defines AmI as “the vision of a future in which the environments support the people inhabiting them. For example, instead of using mice, screens and keyboards, we may communicate directly with our clothes, household devices,...” The identified key features of AmI are: embedment, intelligence, context-awareness, personalization, adaptation and anticipation. It is also mentioned

that “AmI can provide sophisticated support for everyday living, but the information capabilities it may use for this purpose can also potentially provide an invisible and comprehensive surveillance networks – walls literally can have ears”. It inevitably opens up issues of privacy risk, acceptance and security.

In many ambient environments, data arrives as streams or as alerts/notifications and is only relevant for a period of time; its interpretation depends on the user’s context and preferences. For instance, an information about a free parking place can be relevant for a user if this information is recent and if the parking place is nearby the user’s location. Another example is the heat setting to the right temperature in the room where a given person is in and accordingly to his/her own comfort preferences.

In the database community, a lot of work has been devoted to efficiently monitor huge amount of data streams coming from sensors that continuously push their data to a fixed centralized system, without being concerned in privacy, mobility, context-awareness and reactivity. But as soon as a sensor is linked to my personal life (e.g. my home location, my traveling itinerary, etc), the applications using the captured data may become intrusive in my private life. Moreover, in the opposite, as soon as I leave the smart environment, I may lose the ambient capabilities that may support my everyday life (e.g. tension and heart beat measurement, mandatory presence in a certain place). Consequently, the ambient environment and applications are considered as undesirable constraints in some cases and helpful tools in others. Since my ambient environment is changing over the time and over the space (e.g. at home, at work, at the hospital), the query processing should adapt itself to these two dimensions. As Feng said [10] “AmI imposes strong user-centric context-awareness requirement on data management”, but also strong system requirements in terms of hardware constraints (i.e. energy consumption, wireless communication).

As seen before, smartphones and the underlying applications are, under some restrictions, good support for everyday life. However, their repetitive development from scratch is time and money consuming, it makes the software evolution quite difficult, in particular because components updates are frequent. We claim that an embedded data management system for AmI may significantly contribute to ease the development and maintenance of such applications.

The contribution of this paper is to propose an ambient mediation system (called CAIMAN for *C*ontext-aware *d*Ata *I*ntegration and *M*anagement in *A*mbient *e*Nvironments) which :

- facilitates personalized and contextual integration and monitoring of heterogeneous data streams through continuous query execution;
- enables applications to dynamically sense and control, according to some preferences, the ambient environment of the user, which is changing over time and space;
- enables the user to keep some control over his personal data as the monitoring is done exclusively on his personal device.

In the remaining of this paper, Section 2 defines the requirements and the constraints imposed on the design of an ambient mediation system, and presents the architecture of CAĪMAN. In Section 3, the ambient mediation approach is described and illustrated through a scenario. In Section 4, we detail the main components of our system and report on our experimental prototype combining Arduino+Android. Section 5 concludes with some open issues.

2. MOTIVATIONS

To develop ambient applications, there is a need of an ambient data mediation system (ADMS) which allows interoperability between a set of dynamic and loosely-coupled ambient data sources. An ambient data source is a (fixed or mobile) communicating object which generates or consumes continuous (or discontinuous) flows of data. Among such objects, we can distinguish a wide spectrum of sensors and mobile phones as well as any other data services which can push specific data to the applications. In addition to these data sources, there exist other ambient objects called actuators, that do not exchange data, but simply perform some actions on other objects. Notice that a single physical object can play both the role of a data source and actuator. All ambient physical objects are abstracted by software services which encapsulate them and make visible their capabilities, especially their data exchange protocol.

The design choices of our system have been motivated by the requirements of AmI applications in general, and mobile/ubiquitous users/equipments in particular; the key issue being the continuity of ambient services whatever the dynamic changes are. In this section we review them and compare our design choices with existing related work. The proposed CAĪMAN architecture is built on the basis of these choices.

2.1 The requirements

An ambient information system (AmIS) is a set of data flows provided by a collection of ambient objects to achieve the needs of AmI applications (e.g. intelligent home, intelligent city, health care, mobile social network, etc). Some AmIS objects can play the role of a mediator which is able to integrate and interpret data of many ambient data sources, as well as to perform actions over their environment. Most of the AmIS data may persist only a few seconds or minutes in the system, unless the application or the user decides otherwise for various reasons. The main specific requirements imposed to the design of an ADMS are the following :

- Data sources are heterogeneous. They may be fixed or mobile and arbitrarily connect and disconnect from the mediator, during variable intervals of time. Data sources have different capacities in terms of storage and computation.

- The mediator can dynamically connect to the sources when and as long as they are active (i.e. visible over the wireless network and ready to provide data).
- The mediator should provide, for each application, the capability to define its data requirements in terms of event types, so offering a concept of a virtual schema similarly to conventional mediators, and a mechanism which handles continuous queries.
- The mediator should be able to aggregate data flows originated from the same source and integrate data flows originated from different sources on the basis of specific rules provided by the applications. As in conventional mediators, data heterogeneity should be transparent to the user, adaptors are aware of data transformation.
- The mediator should adapt itself to the user's context by continuously searching for the appropriate data sources. It should also satisfy user's preferences in terms of data delivery, relevance to domain of interest, privacy, etc.
- The mediator should be aware of energy consumption and manage consequently the connections to the sources and the usage of its resources.

These requirements clearly distinguish an ambient mediator from a conventional one [21] where the mediation schema and the sources are known in advance. Here the environment is dynamic as data sources enter and leave continuously the field of detection. The personalized and contextual integration and monitoring of heterogeneous data streams rely on continuous query evaluation.

2.2 Related work

In this section, we list the CAĪMAN main objectives and compare them with existing related works.

The first goal of CAĪMAN is to provide a high-level declarative approach which permits user applications to interoperate over distributed ambient objects. The expected data streams are relatively small in their length/size. Many formalisms for event streams processing and querying have been proposed, see [9] for a good survey. In Data Stream Management Systems (DSMS) [4], many CQL-like query languages which extend SQL have been defined. They are based on the concept of window used to manage and filter data streams in a declarative way [5, 14, 8, 13]. In Complex Event Processing (CEP), some formalisms based on composition operators (i.e. sequence, conjunction, disjunction, etc), or time-based automata are used. The goal of CEP is to detect event patterns (i.e. situation) with temporal constraints in data streams. Today, the two approaches are seen as complementary [9, 16]. Both approaches focus on events detections but none on the events reactivity which is an important feature of AmI applications, e.g. the ability to identify the context during which active behavior is relevant and the situations in which it is required. Both approaches assume that the data (i.e. events) are continuously pushed to a centralized system known in advance. These assumptions do not fit with our constraints, as the push mode consume a lot of energy.

In Sensor Databases such as TinyDB [15], data is acquired in a pull mode to avoid battery consumption. The query

(i.e. Tiny SQL) is sent through the network and evaluated in a distributed mode. Sensors are active only when they are queried. The advantages of this approach is its adaptability to the features of hardware devices and to their constraints. The sensor network can contain a large number of sensors. However, the sensors are homogeneous, they all have a TinyOS and there is no mechanism of source discovery because the sensors are all known in advance.

The second goal of CAÏMAN is to make the ADMS aware of the user's context and user's preferences. Again, a lot of work has been devoted to context and preference-aware queries by the database community. Traditionally, the integration of context and preferences in queries is made in two ways [10]. The query pre-processing consists in enriching the query with context or preference informations before executing it. The query post-processing ranks the query's answers according to these informations. Unfortunately, this mechanism works only for one-time queries and not for continuous queries in which the notions of pre and post-processing do not exist. Indeed, in traditional DBMS, data are permanent and queries are transient. In DSMS, data are transient and queries permanent as they are continuously evaluated over the transient data. To our knowledge, no solution has been proposed to handle the context and the user preferences on continuous queries.

In existing context-aware frameworks [6], the context manager is generally represented by a centralized server which is in charge of collecting context information, interpreting and providing them to the client applications. However in pervasive environments, there are frequent disconnections and low connectivity, making this architecture not robust enough and adequate for this type of application. In the literature, only few systems [7, 11] have proposed a local context server to overcome this problem. However, in [11], the context sources are known in advance and correspond to built-in sensors. Conversely in [7], the authors have developed a sentient object model for ad-hoc mobile environments where the context is only used to adapt the application behavior. It doesn't allow to enhance application data with contextual information. For instance, many applications need to add the location to the data produced.

2.3 The CAÏMAN architecture

The overview of the CAÏMAN architecture is depicted in the Figure 1. The resource discovery component facilitates objects discovery and handles dynamic connections and disconnections to these objects. AmIS objects should be able to rely on their own battery, so short-range wireless communication such as Bluetooth are assumed in CAÏMAN as these personal area networks are known to have a low consumption of energy. Once, a data source is discovered the data collectors are responsible for acquiring the data. Data sources do not push their data continuously, but rather sporadically in response to the mediator request. This request is done only if the data source can serve the needs of the applications which have been deployed on top of the mediator. The originality of our ADMS is to offer an hybrid approach combining both the push and the pull modes.

In our environment, we assume that sensors/actuators remain passive most of the time with a default behavior unless someone requests a service (i.e. light off, that can be turned on). All sensors/actuators implement some generic functions (e.g. services) and some that are optional. Sen-

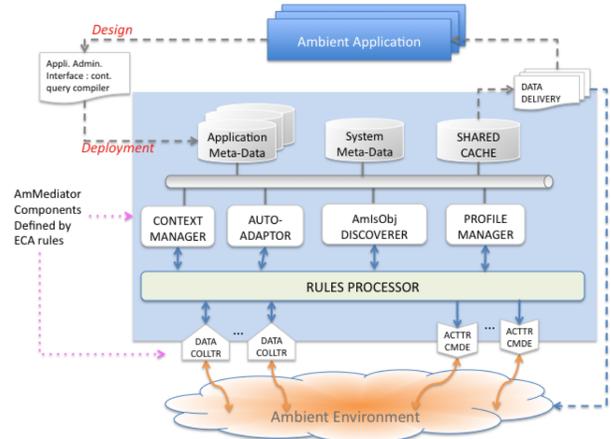


Figure 1: CAÏMAN

sors can only send data during a period of time fixed by the mediator, enabling the sensors and actuators to fall automatically asleep when they have finished their duty and thus turn back to their default state.

As the mediator should fit into lite clients such as smartphones and function in a complete autonomy, no system functionality is delegated to a central server. We don't rely on a global data source registry that might not be available at all time. Meta-data exchange between AmIS objects should be done instead at runtime and the context and profile managers should be local.

CAÏMAN provides a declarative language which allows to describe most of the system and application semantics which is based on the ECA (Event-Condition-Action) paradigm used in active databases [17]. Thus, the rule processor is a core component of our system. It will be detailed in the next section. Notice that in this paper our goal is not to propose yet another query language nor a complex context-aware model, but rather to select a subset of existing formalisms, keep them simple and tractable as much as possible to fit into lite clients.

3. THE AMBIENT MEDIATION APPROACH

In this section, we detail our mediation approach. First, we describe the different types of ambient sources on which the CAÏMAN is built on, then the virtual mediation schema is presented. To understand the approach, we illustrate it with a scenario. In this scenario, Paul is a student and lives at the university residence. He wants to benefit of an intelligent home behavior, by automatically controlling the air conditioning of the room where he is located according to his preferences. He also needs to organize his evenings and wants to be notified about interesting cultural events located not far from his current location. In order to do so, Paul will have to deploy two AmI applications which have been specified in a declarative manner by some designers. During the deployment, the declarative description will be used to instantiate the virtual schema of the mediator, i.e. the application meta-data as described in the Figure 1.

3.1 The Ambient Sources

Three types of data sources [12] are considered: (1) physical sources (e.g. GPS built-in sensors, smartphone, external temperature sensor such Arduino), (2) virtual sources (e.g. user, agenda alerts, SMS, emails, contacts) and (3) logical sources which combine physical and virtual sources with

information from databases. These sources can be either fixed (e.g. already embedded in a mobile device where the mediator is), or dynamic (i.e., another smartphone, a sensor/actuator). Fixed ambient sources are known in advance and always connected to the mediator, e.g. built-in sensors.

Dynamic ambient sources correspond to sources that appear and disappear to the mediation system over time due to the source mobility itself or due to the mobility of the user which embeds a mediator on his personal smartphone. If a smartphone is close to a mobile device, a communication can be established and some messages can be exchanged as long as the device is reachable. If suddenly it disappears due to the user mobility, some messages can be lost. Moreover, the user himself can be an ambient data producer as he/she behaves like any other sensor (intelligent sensor). For instance, the user is discovering a broken window and wants the mediation system to inform automatically the technical staff in charge of repair.

In the remaining of the paper, we focus more on the dynamic data sources. Each one exports its capabilities (e.g., metadata) in an XML document as depicted in the Figure 2. Each dynamic source corresponds to a physical device characterized by an 'id', a 'type' (e.g., Arduino, Android) and a version number.

```

SOURCE 1 : (SENSOR)
<Metadata>
  <Physical id=20 version=2.1 type= Arduino></Physical>
  <Sensor type= Temperature location=Room304 frequency=1000>
</Sensor></Metadata>

SOURCE 2 : (ACTUATOR)
<Metadata>
  <Physical id=30 version=2.2 type= Arduino></Physical>
  <Actuator type= AirConditioning location=Room304 >
    <action name= on ><parameter name=degree></parameter>
    <action name=off ></action> </Actuator>
</Metadata>

```

Figure 2: Sources Description

3.2 A declarative approach

The declarative approach used in CAÏMAN is ECA. ECA rules are a generalization of several methods to achieve active behavior, such as triggers and production rules. ECA rules are evaluated in three steps: (1) when an occurrence of an *event* is detected, (2) the system evaluates the *condition* under which the event is considered relevant, and (3) if it is verified, the rule *action* is executed. The separation between E-C-A is important for many reasons and has been emphasized by the active database community [17].

When designing a mediator based on the ECA paradigm it is important to carefully take into account the life cycle of a rule and the dimensions related to its semantic execution. Indeed this knowledge is mandatory for those designing an application. Moreover by separating the dimensions, there is more flexibility, different behaviors can be proposed for specific applications. In the Figure 3, we summarize them. Here we shortly explain some of the dimensions. The *event detection and composition* and the *visible DB states* will be explained in the next section.

There exist many modes of *event consumption*, among them we selected two modes more suitable to our environment:

1. **recent**: only the most recent instances of any event are considered; older events are discarded. It is most suitable for fast-changing environments in which new events supersede old ones.

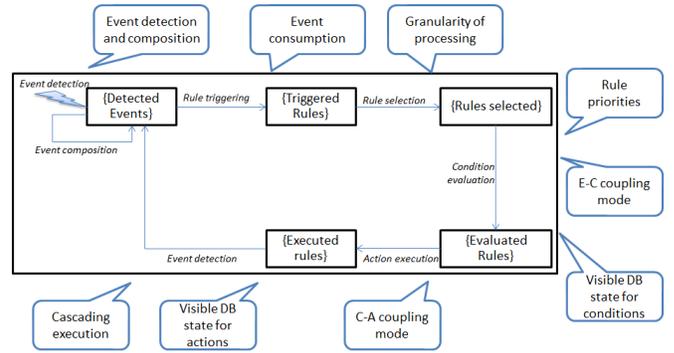


Figure 3: Active Rules & their Execution Semantics

2. **chronicle**: the oldest instances are considered and then discarded; i.e. events are consumed in a chronological order. It is preferred when there is a causal dependency between events.

The *granularity of processing* defines whether the rule processor reacts after the detection of each event (instance-oriented processing) or after a detection of a set of events (set-oriented processing). An example of instance-oriented processing is to call emergency after detecting a critical situation like the unconsciousness of a person. In order to avoid a false alarm, we can wait for multiple events before calling the emergency. In CAÏMAN, this latter dimension can be offered by defining windows on the events arrival. The *rule priorities* determines how rules are selected among a conflict set of rules (i.e. rules triggered by the same event). The EC and CA coupling modes indicates when condition (resp. action) is evaluated after event detection (resp. condition evaluation). The different options are: immediate, deferred, or detached. CAÏMAN proposes immediate coupling modes. In our system, we do not consider *cascading execution* because we assume that our actions have no side effect.

In our experimental prototype, only one semantics is implemented for the rule processor. Rules are evaluated in parallel and no cascading executions is allowed, but the event consumption and the granularity of processing can be parameterized.

3.3 The Ambient Mediation Schema

The CAÏMAN mediation schema is composed of: (1) a set of events types, corresponding to the data flows required by ambient applications, and events detectors, (2) a context model and a default user profile, and (3) a set of personalized and contextual continuous queries (defined as ECA rules).

3.3.1 Events & Event Detectors

An event type can be either simple (*SE*) or complex (*CE*). A complex event type is a combination of other simple or complex events types. These event types are defined by the application designer.

Each *event type* (*SE* & *CE*) is defined by a set of attributes:

- *name*: name of the event type,
- *lifespan*: default time interval during which the event instance is valid,
- *aggrFunction*: function which aggregates events to produce a complex event. For simple events, there is no aggregation function.

Each event type can be instantiated at runtime according to data flows arriving to the mediation system. These *event instances* (*SE & CE*) are defined by a set of attributes:

- *value*: event instance value,
- *source*: source name that captured the event instance,
- *raisingDate*: moment when the event instance is produced/observed by its source,
- *systemTime*: moment when the event instance is detected by the mediation system,
- *lifespan*: time interval during which the event instance is valid after its raisingDate,
- *raisingLocation*: geo-location where an event instance is produced/observed by its source.

The *lifespan* is a metadata which can be provided by the event source or assigned by the application. Event instances are relevant during a limited period of time. Pervasive environments can cause delays between the raising date of an event instance and the time for treating this instance. Consequently the validity V of an event e_i is defined by:

$$V(e_i) = \begin{cases} 1 & \text{if } \text{raisingDate}(e_i) + \text{lifespan}(e_i) < \text{currentTime} \\ 0 & \text{otherwise} \end{cases}$$

The *raisingLocation* is a useful notion for many location-aware applications. Indeed, the location can influence the relevance of a given event instance. For example, an event “flood” detected far away from a user can be irrelevant for him.

Once event types are defined, one should specify how and when event instances are created or captured. This is done by specifying event detectors. Depending on the event type and on the target data source, an event detector may be defined in various ways: a listener, a lookup function or any other procedure able to transform a specific signal into a semantic event.

For our scenario, the designers have separately defined three simple event types : *UnvalidTemperature*, *UnvalidHumidity* and *Advertisement*, with respectively a default lifespan of 5 min, 5 min and 1 week, and one complex event type *UncomfortableSituation* with its associated aggregate function *Foo* as depicted in the Figure 4. For each event, the designer must define a detector. Here, we only give the simple event detector *DT* on temperature expressed as a CQL query and the complex event detector *US* expressed as a CEP-like manner. Others are omitted as they can be expressed in a similar way.

3.3.2 Context Model

According to [20], there exist six different models for representing the context information. Some models like the ontology-based model is very expressive and allows powerful context processing. However in CAÏMAN the model used is the simple key-value model as it should be embedded.

Following our previous work [3], we define a **context** by five dimensions : spatial , temporal, environmental, equipment and user state.

1. The *spatial dimension* is an important characteristics of mobile and pervasive environments. Indeed, depending on how much the user is mobile, the system will react differently. For instance, if a user is highly

```
SimpleDetector DT (EventType : 'UnvalidTemperature', Source : 'Temperature'){
  SELECT raise UnvalidTemperature(
    value : t.value, source : Source.name,
    raisingDate : t.timestamp, systemTime : SYSDATE(),
    raisingLocation: Source.location)
  FROM Temperature t [NOW]
  WHERE t.value not between
    (UserPref.Temperature.min, UserPref.Temperature.max);}

aggrFunction Foo (UnvalidTemperature t, UnvalidHumidity h) {
  raise UncomfortableSituation (
    value= (t.value or h.value); source = 'mediator' ;
    raisingDate = systemTime = SYSDATE();
    raisingLocation = Context.Spatial.locality);}

ComplexDetector US ( 'UncomfortableSituation',
  Foo('UnvalidTemperature', 'UnvalidHumidity'){
  (UnvalidTemperature OR UnvalidHumidity) within 50s;}
```

Figure 4: Simple & complex event detectors

mobile, he will not have time to establish proper connections with all equipments around him. The other important aspect is the location. It can be expressed in many ways depending on the application: GPS coordinates, an address, or a locality label (e.g. Room 305B, Administration Building). The spatial information can be provided by GPS built-in sensors or mobile networks or derived from Google Maps or databases.

2. The *temporal dimension* is an important information that can be used for personalizing an application. For instance, a user can be interested to receive events only in the morning. Designers can change the notion of moment in the core context, e.g. when the morning begins and ends. This information can be provided by the phone clock (i.e. date, time) or derived from context definition (i.e. moment).
3. The *environmental dimension* concerns all the sensors describing the environment of the user (e.g. temperature, humidity, luminosity). This information is important when the environment is fixed, for example a smart home.
4. The *equipment dimension* characterizes all information about the media used by the user to interact with the application: the used device (e.g. type, battery autonomy, memory storage, computing power) and the connectivity (e.g. type of connection, rate). This dimension is important to adapt dynamically the system accordingly with the equipment constraints (e.g. low battery, uncertain connectivity).
5. The *user state dimension* allows to know if a user is available or not, and how he feels. In our case, we are more interested in the user availability which can have an impact on the system behavior.

Designers have to provide the context model used by their application and the set of context rules to the mediator that can compute the current context. For that, a list of context models is proposed with default context rules. For example, if the application designer is interested in the location information, there exists a rule associated with this dimension which captures the GPS coordinates from the smartphone GPS sensor every minute. However designers can also write their own context rules and submit them to the mediator.

3.3.3 Profile Model

The survey [19] highlights the difficulty of choosing the good representation for the user preferences: qualitative or

quantitative. The quantitative approach allows a total order between preferences but is not intuitive because this implies that the user put weights on his preferences. While the qualitative approach is very intuitive but makes difficult the usage because there is not necessarily a total order between preferences. In CAIMAN the user preferences considered are viewed as dynamic criteria by the application. Thus there is no order between preferences as all preferences must be considered by the application.

Following the definition given in [3], a **user profile** is organized into several dimensions, possibly decomposed into sub-dimensions. Each dimension and its sub-dimensions contain a set of attributes and their values on which preferences are expressed. We retain four important dimensions:

1. *Personal data* contains all information about the user (e.g. his name, his address, his birthday). This dimension may also contain data on social groups to which the user belongs to (e.g. student, professor).
2. *Domain of interest* is generally the central dimension of the user profile, it represents the user domain of interest and preferences. For example, the domain of interest may be the types of events the user is interested in and the preferences on how/when event instances are received or treated.
3. *Resource discovery* contains the user preferences on the remote resources (i.e. type of resource, associated data collectors, related security issues and all meta data useful to understand data stream semantics). An example can be receiving events only from sources located in the same place as the user.
4. *System adaptation* groups user preferences on how the system should adapt to the user context or to its own behavioral parameters. An example can be to disable the resource discovery component during the night. A set of preference rules can also be defined to adapt the system behavior in case of low battery, that is either disabling some functionalities or changing the frequency of the captured data.

In the same way as the context, designers specify a default user profile for their application. The Figure 5 describes the application default profile set by the designers for our scenario. We have gathered the profile of each application into a global one, but in reality each application has its own. $\$name$ and $\$\$name$ variables represent respectively a string or a set of strings.

```

Profile
Resource Discovery :
  sensor.location = Context.Spatial.locality;
  actuator.location = Context.Spatial.locality;
DomainsOfInterest :
  on Temperature t:
    if( Context.Temporal.Time between(8.00,20.00) )
      then { min = $1, max = $2 } else { min = $3, max = $4 };
  on Advertisement a:
    condition : (a.value.topic in $$5)
               AND distance(Context.Spatial.Locality, a.raisingLocation)<$6);
End Profile

```

Figure 5: User Profile

For each application, the designer has specified the resource discovery constraints. Indeed, to control the temperature in a room, we are only interested by sensors and

actuators located in the same room where the user is currently located. Some variables have a default value set by the designer. If not, they will be filled by the user when installing his application on his mobile device. For instance, the default temperature variables are ($\$1=18^\circ$, $\$2=22^\circ$) for the day time and ($\$3=16^\circ$, $\$4=18^\circ$) for the night. In the same manner, advertisements relevant to a user are those within $\$6=15\text{km}$. Others variables are set by the user. For instance, Paul decides to change the default value $\$6$ to 25km , and set the $\$\5 to ('music', 'sport').

3.3.4 Application ECA Rules

The last task of the designer consists in defining his set of ECA application rules. For example, let's consider the following rules defined in the Figure 6 that model our scenarii. As said before rules are contextual and personalized. Thus, the context and the user profile can be either explicitly used by designers in their rules and in particular in the condition, or implicitly used by the system during the runtime when specified within the user profile.

The first rule of the smart office scenario explicitly uses the user profile and in particular the UserPref.Temperature.min and UserPref.Temperature.max variables described above. As these variables are context-dependent (i.e. day or night), their values are changing over time. So, they will be instantiated just before evaluating the rule condition.

In the second rule, a user receives advertisements. Here no condition is specified; however one will be added by the system at runtime since an implicit preference has been defined on this event type in the default profile. The reason why the condition is not directly written in the rule, is to allow the user to change his condition at any time. Here, users receive advertisements relevant to their personal topics and within the required distance. When an event is relevant for the user, he is informed by email.

```

Define ECA_Rule Application_SmartOffice_R1
  EVENT : UncomfortableSituation u [range 5 min]
  CONDITION : AVG(u.value) not between
              UserPref.Temperature.min and UserPref.Temperature.max
  ACTION : activate ( 'airConditioning', 'on', UserPref.Temperature.min+2)

Define ECA_Rule Application_Culture_R1
  EVENT : Advertisement a [Now]
  ACTION : inform( User, 'mail', a)

```

Figure 6: Rule Examples

Up to now, we have given some intuitions of our rule language, now the semantics of rules is briefly described.

The granularity of processing (e.g. instance or set-oriented) is defined as a window. The default window [Now] is a time-based window of size 1 and corresponds to the instance-oriented. Others windows are defined as in any continuous query language and correspond to the set-oriented. For instance, "*range 5 min*" represents all events that appear within the last five minutes. Notice that we do not allow unbounded windows. The visible state of conditions and actions is defined by the window and the event type. The event consumption mode is defined as a fixed parameter of the mediator and is taken into account in event detectors that specify how to raise events.

Some generic actions are possible and are defined by a template. The first action "*inform(\$who, \$show, \$what)*" informs a user or a group of users, or an application through a

delivery mode (i.e. local, wireless, email, SMS), of a particular message or event. We separate messages from events, messages are generated to users and definitively leave the system, by opposition to events that will be filtered and re-injected later in a mediator that can interpret them. The action “*activate(\$type, \$service, \$serviceparameters)*” activates a service with some parameters on a specific actuator type that satisfies some user preferences.

4. THE CAÏMAN MEDIATION SYSTEM

In this section, we briefly describe the behavior of the main components of our system at runtime. Thus, we assume that AmI applications have been deployed on the smartphone, and that their default profile, detectors and rules have been transmitted to the mediator and uploaded in the application metadata database.

4.1 Data Collectors & Actuator Commands

As the data sources are heterogeneous, a set of generic data collectors and actuator commands are needed to allow communications with the mediation system. The *Data Collectors* are used to collect and transform any kind of heterogeneous data so it can be understood and integrated in the mediator. Each source is bound with one instance of a data collector that is responsible for querying and controlling this data source. All communications are asynchronous. Another issue also considered is the data transformation as the data provided by a source is not necessarily compatible to the coding, format, unit and scale of the expected data at the mediator level. The data collector is in charge of transforming these raw data into events, thus it activates the simple event detector associated with the source.

Actuator Commands allows the mediator to transform commands into real actions on the environment through actuators services.

4.2 Resource discovery & Bindings

Due to the numerous communicating objects that enter and leave the field of detection because of the user mobility, there is a need of a *Resource Discovery* component which is continuously aware of the equipment's environment.

Ambient data sources may connect and disconnect arbitrarily. As the mediator cannot rely on a centralized resources registry, the resource discovery service is defined as a seeking function which detects the surrounding objects, identifies them through some metadata exchange and establishes connections to them if they are relevant at least for one active application deployed on the mediator.

As connections/disconnections can be frequent, the resource discovery component stores a history of all the discovered sources with their version number. This version number is useful to know if the source has changed since the last time it was discovered. This is to avoid unnecessary metadata exchanges. The history plays the role of a local resource registry which can be deleted at anytime, since it can be rebuilt at runtime.

Before communicating with a data source, one should know if the information it contains is relevant. For doing so, a matching is made between the source metadata and a part of the mediation schema. At the same time the context and profile information is used to select only the sources in

a given location and at a given time, according to the preferences of the application or the user. Once the source is selected and a communication established, a dynamic binding enables to link the data source to the mediator, by instantiating the suitable data collector.

4.3 Profile & Context Manager

As many context-aware systems [6], the mediation system proposed makes a separation between the context acquisition, the context processing, and the context information usage. In fact, data collectors are in charge of retrieve context raw data, the context manager processes these data to infer context information which will be stored and used by applications and in particular application rules. These context information are also used by the profile manager to derive the active profile (i.e., all profile information which is valid for this current context). Indeed, the user profile is composed of a static part and a dynamic part. In the first one, the profile information doesn't change frequently while the second one depends on a context that can change rapidly. As we have seen in the Figure 5, the user profile can be contextual and it will be the role of the context and profile managers to keep the active profile up to date for the application. For simplicity reasons, an assumption is made on defining a contextual profile. Only If-then-else statements are allowed in order to avoid conflicts between contextual predicates. Only one active profile is valid at any instant of time. Notice also that all variables can be changed at any time by the user, via a simple interface on his smartphone.

4.4 Rule Processor

The *Rule Processor* is an idempotent service to which ECA rules with their associated detectors are submitted. As said earlier, it is important to follow rule execution semantic as described in the Figure 3. The query processor relies on a multi-threaded execution framework. The approach we follow is, in a way, very similar to what has been proposed by Krämer [14] and especially their SweepArea that models a dynamic data structure to manage a collection of events of the same type. ECA rules act as continuous queries over collection of events and react over their environment when a situation is encountered.

As events of the same type can be used in many rules, events are not removed when used for triggering a rule. Thus, a garbage collector is necessary and events are removed from the dynamic structure when their lifespan has expired. In order to avoid triggering multiple times a rule with the same events, each rule has a context summarizing the past execution. When the rule processor is looking for the rules that can be triggered, it uses the context which is also computed in a continuous way. Only the most recent event is kept for each dimension.

4.5 The prototype

Smartphones as well as computers cannot really sense the world. For AmI environments, there is a need for tools for making computers that can sense and control more of the physical world than any other desktop computer. This is the role of the Arduino [2] platform to sense the environment by receiving input from a variety of sensors and to affect its surroundings by controlling lights, motors, and other actuators. A first prototype combining Arduino and Android validating the approach has been implemented. Many sensors and actuators have been prototyped. The CAÏMAN mediator and

many simple AmI applications can be deployed on an ANDROID platform. For the time being, data collectors and actuators are operational, as well as the resource discovery. Simple and complex event detectors, as well as simple ECA rules are supported. When the validity of events expires, the garbage collector automatically deletes the events. We are currently integrating the context and the user profile within the rules. The user agrees or not to install an AmI application on top of CAĪMAN. He can turn it on/off whenever he wants to. He can also checked the types and the number of events, the mediator is currently monitoring.

5. CONCLUSION

In this paper, we have presented the different requirements posed by ambient environments and proposed an ambient mediation system called CAĪMAN. As we have seen, a declarative approach based on ECA rules is proposed. The main originality is that it combines in an elegant way the context, the user profile and the continuous queries together. Some dimensions of the profiles can be integrated and changed at anytime by the user. Another important contribution of our work is that it is based on personal mobile devices and local computation that better fulfill the user privacy. To our knowledge CAĪMAN is the first ambient mediation system embedded in a smartphone offering such functionalities.

Some open issues still remain to be considered such as how to adapt the system if the context is critical (i.e., low battery), what to do when many mediators are acting in a conflicting way on specific resources, how an event that has been sent several times by different data sources can be discovered to avoid repeating the same actions again and again.

Acknowledgment

This work is partially funded by the French National Agency for Research (ANR) project under grant KISS (2011-INSe-005-03).

6. REFERENCES

- [1] ABI research. <http://www.abiresearch.com/press/1466-In+2014+Monthly+Mobile+Data+Traffic+Will+Exceed+2008+Total>.
- [2] Arduino site. www.arduino.cc.
- [3] S. Abbar, M. Bouzeghoub, D. Kostadinov, S. Lopes, A. Aghasaryan, and S. Betge-Brezetz. A personalized access model: concepts and services for content delivery platforms. In *Proc. of the 10th Int. Conf. on Information Integration and Web-based Applications and Services.*, pages 41–47, New York, USA, 2008.
- [4] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein, and J. Widom. STREAM: the Stanford stream data manager (demo). In *Proc. of the ACM SIGMOD Int. Conf. on Management of data*, pages 665–665, New York, USA, 2003.
- [5] A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2):121–142, 2006.
- [6] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *Int. J. Ad Hoc Ubiquitous Comput.*, 2(4):263–277, 2007.
- [7] G. Biegel and V. Cahill. A framework for developing mobile, context-aware applications. In *Proc. of the 2nd IEEE Int. Conf on Perv. Comp. and Comm. (PerCom'04)*, pages 361–, Washington, USA, 2004.
- [8] I. Botan, R. Derakhshan, N. Dindar, L. Haas, R. J. Miller, and N. Tatbul. SECRET: a model for analysis of the execution semantics of stream processing systems. *Proc. VLDB Endow.*, 3(1-2):232–243, Sept. 2010.
- [9] M. Eckert, F. Bry, S. Brodt, O. Poppe, and S. Hausmann. A CEP Babelfish: Languages for Complex Event Processing and Querying Surveyed. In S. Helmer, A. Pouloussis, and F. Xhafa, editors, *Reasoning in Event-Based Distributed Systems*, volume 347 of *Studies in Computational Intelligence*, pages 47–70. Springer Berlin / Heidelberg, 2011.
- [10] L. Feng, P. P. M. Apers, and P. W. Jonker. Towards context-aware data management for ambient intelligence. In *15th Int. Conf. on Database and Expert Systems Applications*, pages 422–431, 2004.
- [11] T. Hofer, W. Schwinger, M. Pichler, G. Leonhartsberger, J. Altmann, and W. Retschitzegger. Context-awareness on mobile devices - the hydrogen approach. In *Proc. of the 36th Annual Hawaii Int. Conf. on Syst. Sci. (HICSS'03)*, pages 292.1–, Washington, DC, USA, 2003.
- [12] J. Indulska and P. Sutton. Location management in pervasive systems. In *Proc. of the Australasian inf. sec. workshop conf. on ACSW frontiers*, pages 143–151, Darlinghurst, Australia, Australia, 2003.
- [13] N. Jain, S. Mishra, A. Srinivasan, J. Gehrke, J. Widom, H. Balakrishnan, U. Çetintemel, M. Cherniack, R. Tibbetts, and S. Zdonik. Towards a streaming SQL standard. *Proc. VLDB Endow.*, 1(2):1379–1390, Aug. 2008.
- [14] J. Krämer and B. Seeger. Semantics and implementation of continuous sliding window queries over data streams. *ACM Trans. Database Syst.*, 34(1):4:1–4:49, 2009.
- [15] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.
- [16] A. Margara and G. Cugola. Processing flows of information: from data stream to complex event processing. In *Proc. of the 5th ACM int. conf. on Distributed event-based system*, DEBS '11, pages 359–360, New York, NY, USA, 2011. ACM.
- [17] N. W. Paton and O. Diaz. Active database systems. *ACM Comput. Surv.*, 31(1):63–103, 1999.
- [18] F. Sadri. Ambient intelligence: A survey. *ACM Comput. Surv.*, 43(4):36:1–36:66, Oct. 2011.
- [19] K. Stefanidis, G. Koutrika, and E. Pitoura. A survey on representation, composition and application of preferences in database systems. *ACM Trans. Database Syst.*, 36(3):19, 2011.
- [20] T. Strang and C. L. Popien. A context modeling survey. In *UbiComp 1st Int. Workshop on Advanced Context Modelling, Reasoning and Management*, pages 31–41, September 2004.
- [21] G. Wiederhold. Mediation in information systems. *ACM Comput. Surv.*, 27(2):265–267, 1995.