Integrating Tiny Heterogenous and Autonomous Data Sources

Kim Tâm Huynh
Prism Laboratory,
University of Versailles,
Versailles, France
kim-tam.huynh@prism.uvsq.fr

Béatrice Finance
Prism Laboratory,
University of Versailles,
Versailles, France
beatrice.finance@prism.uvsq.fr

Mokrane Bouzeghoub
Prism Laboratory,
University of Versailles,
Versailles, France
mokrane.bouzeghoub@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. 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 provides facilities to develop and deploy ambient applications over devices such as smartphones. This paper reports on our first experiment prototype, combining Arduino+Android, in using such ambient mediator for an intelligent home application.An embedded mediation system CAIMAN is proposed and illustrated through a simple scenario.**

***Keywords*—ambient data;embedded system;mediation system.**

I. INTRODUCTION

Over the last 20 years, new paradigms such as ubiquitous computing, pervasive computing, ambient intelligence (AmI) have emerged with the development of wireless networks and the miniaturization of hardware components. Augusto and McCullagh [1] characterized AmI as "a digital environment that proactively, but sensibly, supports people in their daily lives". The challenges posed by these paradigms are addressed by several research communities (networks, multi-agent systems, databases, Human-Machine Interface).

Today, we are witnessing an unprecedent explosion of mobile data volumes, i.e., ambient data. In 2011, 1.08 billion of mobile phone users have a smartphone. Smartphones as well as computers cannot really sense the world. In AmI environments, there is a need for tools for sensing and controlling more of the physical world . This is the role of the Arduino platform that can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling lights, motors, and other actuators.

More and more ambient applications are developed for mobile environments, e.g., Waze [2], APILA [3]. Unfortunately, they are often developed from scratch, which is time and money consuming, and makes the software evolution quite difficult, in particular because components updates are frequent. The lack of a data management system for AmI does not ease the development of applications.

Ambient data have specific characteristics, they arrive as streams or as alert/notifications. Moreover, data are only relevant for a period of time and their interpretation depend on the user's context and the user's 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 and accordingly to his preferences. Such data streams are relatively small in their length/size.

For managing and integrating data streams, the database community has proposed two paradigms: DSMS (Data Stream Management System) processing [4] and Sensor Databases such as TinyDB [5]. DSMS is an evolution of the traditional DBMS (Data Base Management System). In DBMS, data are stored and users issue one-time queries on stored data. In others words, data are permanent and queries are transient. DBMS are not suitable for data streams. Indeed in DSMS, data are transient and queries permanent since they are continuously evaluated over the transient data, they are called continuous queries. A language CQL, has been proposed for managing and filtering data streams in a declarative way. Generally, these systems are centralized or clustered and assume that the data sources, i.e., sensors send continuously their data in a push mode, towards the DSMS. This assumption works if the data sources are known in advance, their schema does not change, they are always connected to the ethernet and do not have limited power. Indeed the push mode consumes a lot of energy, that's why sensor databases have been proposed such as TinyDB. In this paradigm, we assume a network of sensors, 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 have to answer a query. The advantages of this approach is that it is well adapted to the specificities of material and 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 known. These two paradigms are not context-aware and cannot take dynamically into account heterogeneous and autonomous tiny data sources.

In this paper, we propose an ambient data mediation system which offers contextual and personalized data integration over autonomous and heterogenous tiny data sources such as

 ISBN: 978-1-61208-235-6

smartphones and sensors/actuators, in a declarative way.

The plan of the paper is as follows. In Section 2, the motivations and the requirements for such a mediator are given. In Section 3, the ambient mediation approach is described. Finally, in Section 4, an application scenario is designed on top of our mediattor to illustrate our approach.

II. Motivations and Requirements

For deploying 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. In addition to these data sources, there exist other ambient objects called actuators, that do not exchange data, but simply perform some actions, e.g., a led. 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.

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, health care. AmIS Objects may communicate between each other based on various communication protocols. For instance, sensors/actuators micro-controllers only offer a Wire-two-Wire Interface (TWI/I2C) for sharing data over a net of devices or sensors. On the other side, smartphones can exchange data in a more elaborate way. 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 decide 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 connected and disconnected 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 similar concept as a mediator virtual schema, and a mechanism, which handles continuous queries.
- The mediator should be able to aggregate data flows originating from the same source and integrates data flows originating from different sources.
- The mediator should adapt itself to the user's context by continuously searching for the appropriate data sources, e.g., depending on the location and the time. It should also satisfy user's preferences in terms of data delivery, relevance to domain of interest, privacy.
- 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 [6].

III. The Ambient Mediation Approach

We are currently designing and implementing an ADMS, called CAÏMAN for Context-aware dAta Integration and Management in Ambient eNvironments. The overview of the CAÏMAN architecture is depicted in Figure 1. Our aim is not to provide a complete set of data management services but rather a limited set of necessary functionalities to support the design of ambient applications that fit into lite clients such as smartphone, and exploit ambient data. The first goal is to provide a high-level declarative approach, based on ECA (Event-Condition-Action) rules/triggers [7], which permits user applications to interoperate over distributed ambient objects. The second goal is to facilitate object discovery and to handle dynamic connection/disconnection to these objects. The third goal is to make the ADMS aware of the user's context and user's preferences. For example, when a battery of a given equipment is low or when a user is too busy and does not want to be disturbed by anything, the rule processor should stop.

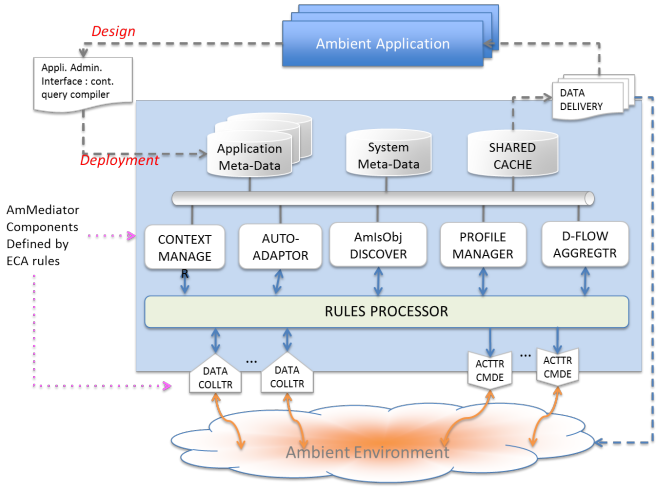

Fig. 1. CAÏMAN

The following subsections give an informal description of the main components of CAÏMAN.

A. The Ambient Mediation Schema

The CAÏMAN mediation schema is defined as a set of events types, corresponding to the data flows required by ambient applications. 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.

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 instance*** *(SE & CE)* is defined by a set of attributes:

- *value*: event instance value,
- *source*: source name that captures 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 and the time of treating this event. The *raisingLocation* is a very 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, the application designer should specify how and when event instances are created or captured. This is done by specifying event detectors with windowing function. 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. Finally, a set of continuous ECA rules is defined.

B. Binding ambient resources to the mediator

In conventional mediators, data sources are known and linked once for all to the mediator at design time. In the context of an ambient mediation system, data sources are not known in advance but dynamically discovered at run time.

Ambient data sources are pervasive services, which may connect and disconnect arbitrarily, hence a centralized catalog of resources is useless. Only active objects in a given context are visible to the mediator. The *Resource Discovery* service is defined as a seeking function, which detects the surrounding active objects and establishes connections to them (called dynamic bindings). Binding a given data source to the mediator consists in matching the source meta data against a part of the mediation schema. If the match succeeds, it means that the data source can provide information to applications running over the ambient mediator, otherwise the remote source is considered as useless. A binding is then defined as a set of contextual mappings. The services provided by the mediator are then dependent on the successful mappings retrieved in the current location at a certain date. One of the main issues of the discovery process is to guarantee a continuous service even if data sources disconnect frequently. Besides the bindings, another issue that should be considered is data transformation. 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. Data transformation is then another important functionality of the mediator. Source binding and data transformation services form what we call a data collector.

C. ECA Rules Processor

Another fundamental service of CAÏMAN is the rule processor. Indeed, one of the main feature of our ADMS is its capability to provide a declarative language, which allows to describe most of the system semantics and the application semantics. This declarative language is the ECA rules language. User applications and mediation services are then defined by ECA rules. Each rule is defined using one or several event types defined in the mediation schema. The rule processor is an idempotent service to which ECA rules are submitted to be evaluated as long as event instances are produced by the application or the mediator. The rule processor has an operational semantics, which is clearly specified by various parameters such as event consumption and coupling modes.

D. Others components

The *Application Metadata* contain event types, ECA rules, the context model, and the default user profile defined by the designer. These information are necessary for the different components. By using the context model, the *Context Manager* computes the current context, which can be used by the *Profile Manager* to infer the active profile, i.e., all user profile information, which are valid for this context.

Concerning the data, once the source is discovered and the data transformed by the *Data Collector*, the mediator proposes a *DataFlow Aggregator* component to process these flows of data and aggregate them. After executing the application rules, the *Data Delivery* component can deliver the result to the application or its ambient environment. The mediator can also execute actions through the *Actuator Command* component.

IV. The Application Scenario

In this section, a specific scenario is chosen in order to illustrate and demonstrate our approach. Let us consider an ambient application scenario that wants to automatically control the air conditioning in the room where the person is, accordingly to his preferences. The user is mobile and can move from one room to another while keeping around the right air conditioning. For simplicity, we assume that the user is alone in the room. For doing so the application is constantly checking its environment to find if the room is well equipped, if it contains either a sensor of temperature or of humidity, and if an air conditioning actuator is present.

We first describe the ambient environment, which is composed of heterogeneous ambient data sources. Then, we illustrate the task that should done by our application scenario designer and what will happen when deploying it on top of our ambient mediation system.

 ISBN: 978-1-61208-235-6

A. The ambient sources

In our ambient environment, we consider two sensors that capture respectively the humidity and the temperature and one actuator for the air conditioning. Each ambient data that is produced has a value and a timestamp, which corresponds to the time when the data has been captured. Each data source captures data at its own frequency. For each source corresponds a physical device characterized by an 'id', a 'type' and a version number. Each ambient source can export its capabilities in an XML document as depicted in Figure 2.

```
SOURCE 1 : (SENSOR)
<Metadata><Physical id=20 version=2.1  type= Arduino></Physical>
  <Sensor type= Humidity 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 type=int name=duration></parameter></action>
   <action name=off ></action> </Actuator></Metadata
```

Fig. 2. Sources Description

B. Design & Deployment

Most of the information that need to be specified by the designer are depicted in Figure 3 and explained in the following. First, the designer defines two simple event types: *UnvalidTemperature* and *UnvalidHumidity*. Both have a default lifespan of 2 min and no aggrFunction. Then, the complex event *UncomfortableSituation,* which is composed of the two simple events, is defined. The default lifespan is 5 min and an aggFunction Foo is associated. For each event, the designer must define a detector. In this scenario, simple event detectors are expressed declaratively in CQL-like manner and complex event detectors use a CEP-like language, composed of operators such as disjunction, sequence, etc. As said earlier, detectors can be defined in various ways.

The simple detector *DT* raises the *UnvalidTemperature* event when the temperature is not acceptable by the user. Due to space limitations, the *UnvalidHumidity* detector is omitted, since it is defined in a similar way. The complex detector *US* raises the *UncomfortableSituation* event when one of the simple event is raised within 50s. It uses the Foo aggrFunction for computing the event instance values. Notice that since data sources do not provide a lifespan, all detectors use the default value defined earlier by the designer.

For his application, the designer only needs the locality for his context model. For simplicity, we assume that the mediator already provides the detector function for this context. The contextual preferences of the user state that only sources and actuators that are located in the same room where he is, are accepted. So the designer provides the default profile for the application scenario by defining the resource discovery policies that are contextual. He also gives the domains of interest preferences such as the min and max temperature. Default values of the profile can be changed by the user at any time. Finally, the ECA application rule *MyScenario* is expressed, it consists in detecting an uncomfortable situation for the user and activating the air conditioning.

Once the event types, the detectors and the ECA rules are given, the application is compiled and deployed over the

```
Define 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);}

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

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

Profile
        Resource Discovery :
                source.location = Context.Spatial.locality; actuator.location = Context.Spatial.locality
        DomainsOfInterest :
                UserPref.Temperature : min = 16, max = 22; UserPref.Humidity : min = 20%, max = 40%
End Profile

Define ECA_Rule MyScenario
        EVENT : UncomfortableSituation[Now]
        ACTION : Activate ('airConditioning', 'on', '10 min')
```

Fig. 3. Design Phase

mediator. Once the application is started, the mediator creates the execution environment for the application. It activates the rule processor with the relevant ECA rules, as well as the complex event detectors. Once a relevant source is detected by the resource discovery component, a data collector is instantiated. It is responsible for the dynamic bindings. For instance, the right adaptor i.e., Arduino, is selected. All simple event detectors corresponding to the type of the data managed by the source are activated, e.g., temperature, humidity. The data collector requests data from the source. Then, event instance streams enter the mediator and are processed. When a source disappears the mediation removes the data collector instance, which in turn deletes all unnecessary flows and simple event detectors associated to the source.

V. Conclusion and Future Work

In this paper, we have presented the requirements of AmI applications and proposed a mediation system CAÏMAN. Our system has been illustrated with a simple scenario. Its goal is to facilitate the development of embedded applications on mobile devices, that integrate ambient data which are different from conventional data. The approach is declarative. Its originality is to take into account, during the rules evaluation, the context and the user preferences. Application rules can be parameterized by a user so as his smartphone could be adapted to his personal needs. Ambient sources are fully implemented on Arduino boards and export their XML capabilities. The mediator is still under development on an Android smartphone. Performance evaluation still remains to be done.

References

[1] J. C. Augusto and P. J. McCullagh, "Ambient intelligence: Concepts and applications," *ComSIS*, vol. 4, no. 1, pp. 1–27, 2007.

[2] (2012) The Waze website. [Online]. Available: http://www.waze.com/

[3] (2012) The Apila website. [Online]. Available: http://www.apila.fr/

[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 description)," in *Proc. SIGMOD'03*, 2003, p. 665.

[5] 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.*, vol. 30, no. 1, pp. 122–173, 2005.

[6] G. Wiederhold, "Mediators in the architecture of future information systems," *Computer*, vol. 25, no. 3, pp. 38–49, Mar. 1992.

[7] N. W. Paton and O. Díaz, "Active database systems," *ACM Comput. Surv.*, vol. 31, no. 1, pp. 63–103, 1999.

 ISBN: 978-1-61208-235-6