Scalable Computing: Practice and Experience
Volume 11, Number 1, pp. 19–32. http://www.scpe.org
ISSN 1895-1767

SENSORS DATA-STREAM PROCESSING MIDDLEWARE BASED ON MULTI-AGENT MODEL*

OVIDIU ARITONI† AND VIOREL NEGRU‡

Abstract. The goal of this study is to propose an architecture for an intelligent sensor data processing middleware. In order to fulfill the ambient assisted living data processing requirements we design a flexible and scalable architecture based on multi-agent model. This architecture allows acquisition, interpretation and aggregation of sensor data-streams. Our system is able to process different sensor data-streams, to adapt to different levels of abstraction, to define different data-processing workflows. An extended operators over data stream language is used to define workflows. Different types of agents (simple sensor agent, logic sensor agent, virtual sensor agent etc) are defined. The designed system is a domain independent multi-agent system which can be instantiated for particular AmI problems. The middleware architecture will use a healthcare system for validation.

Key words: ambient intelligence, ambient assisted living, context-aware sensitivity, healthcare systems, multi-agent model.

1. Introduction. The goal of an Ambient Intelligence (AmI) system is to response in an intelligent way to changes of context, providing intelligent services and adaptation to changing after the delivery [5]. These services include the control of the actuators integrated in the application, but also the acquisition of contextual information. AmI integrates a set of concepts from mobile computing, intelligent sensor, artificial intelligence, service oriented architecture and reflexive and adaptive systems.

1.1. Ambient Assisted Living Systems. Ambient Assisted Living (AAL) systems provides services support for daily life based on context and the situation of the assisted person.

The healthcare context awareness systems are a distinct category of AAL systems. These systems are used in hospitals or houses to improve the patient's life and to provide some useful services for some specialists, such as the nurses and doctors, for a rapid intervention in the patient life.

In [6] is described a healthcare system which provides an intelligent bed that eases the patient life. This system knows the patient's or nurse's identity and displays some relevant information which depends on the person which accesses them. This system is also used to store the electronic records of the patients.

The system described in [28] uses another perspective in healthcare systems: helps the emergency team on their intervention in different situations. The application is used for the localization of the emergency team members, via some active badges, and the communication between them. The potential emergency team members are notified by the application to join the team and if one member is not reachable the system searches another doctor or nurse. Also if an emergency team member is reachable but is not able to join the team it can initiate an audio-video conference.

The MobileWARD project [21] describes an architecture to support the morning procedure from a hospital, being able to display a patient list and all the patient information. De facto MobileWARD project develops a prototype used as an electronic patient records database with some context-aware facilities.

An ubiquitous system used for home medication monitoring and assistance is presented in [12]. A prototype is proposed that uses the RFID technology for detecting the pill bottle position and weight. Another similar project [13] uses the mobile communication to send reminder messages to the patient and information about the patient medication to his doctor.

VirtualECare [11] is an agent oriented assisted living project that uses web services to provide resources management or monitoring facilities. A similar project that use a service-oriented approach is AMIGO [38].

1.2. Article Overview. The main goal of this article is to propose a middleware architecture that allows acquisition, interpretation and aggregation of sensor data-streams. These operations over data must be deployed at the sensors middleware in order to solve the rapid response challenge.

The sensor data and the operations over them are used for:

- directly and immediately alerting doctors and paramedics to attend the patient. This includes in some situations reasoning and interpretation over data-streams.
- storage and building some historical data-streams for diagnosis and future research;

*This work is partially supported by the FP7 project DEHEMS and PNII national project ASISTSYS.
†IeAT - E-Austria Institute Timisoara, România, (oaritoni@info.uvt.ro).
‡West University of Timisoara, Department of Computer Science, Timisoara, România, (vnegru@info.uvt.ro).

- building new data-stream using the received data and techniques such as completing or decompleting data-stream, event detection, inflexion point detection etc. (For details see section 3.6.1)
- detect anomalies or irregularities in the received data.

In order to fulfill the AAL data processing requirements we design a flexible and scalable architecture based on multi-agent model. Our system is able to process different sensor data-streams, to adapt to different levels of abstraction, to define different data-processing workflows.

We define different types of agents (simple sensor agent, logic sensor agent, virtual sensor agent etc). The designed system is a domain independent multi-agent system which can be instantiated for particular AmI problems.

The rest of this paper is organized as follows: the *section 2* presents the general requirements for sensor middleware oriented to AAL systems; the *section 3* describes the proposed architecture; the *section 4* describes the related work and the advantages of the proposed architecture; the *section 5* will conclude the results presented in this paper, and offer an overview of future research.

2. Requirements. The section presents the main problems occurred in the design and development of a healthcare context-aware sensor middleware. The design process of such systems must take in consideration many aspects: the design of wireless sensor network (WSN), the data models, the communication protocol between simple sensor node and the sink node, the middleware runtime support, the middleware services, the quality services mechanisms, the data management (including data acquisition, data processing and data storage), the information discovery and the resource management, the middleware support for interoperability with another systems, the capabilities for software auto-reconfiguration.

In order to respect the interoperability requirement all the received data must respect imposed standardization. There are ISO standards that describes biomedical data such as ISO / IEEE 11073-30300:2004 (that defines an IrDA-based transport profile for medical device communication) [2], ISO 13606-3:2009 (for the communication of part or all of the electronic health record (EHR) of a single identified subject of care between EHR systems, or between EHR systems and a centralized EHR data repository) [4], ISO 12967-1:2009 (provides guidance for the description, planning and development of new systems, as well as for the integration of existing information systems) [3] . The sensor middleware must validate the received data using these standards. Also, to improve the application flexibility, the middleware allows the specification of new standards.

WSNs are constrained by resources such as: bandwidth, computation, communication capabilities, energy, etc. The middleware is used to provide an efficient management of all wireless sensor node.

There are three approaches to implement data-storage [32, 37]: external-storage, local-storage and data-centric storage. The data centric storage is the most popular approach. The external approach is used to store the data in a database station outside the sensor network and the local approach for storing the data where they are generated. The data-centric storage is a compromise between these two approaches.

3. Proposed Architecture. This section describes the proposed architecture for the context-aware sensor data-stream middleware. The solution regards the improvement of flexibility and scalability of such middleware. The proposed design tries to fulfil the ambient intelligent and the healthcare context-aware systems requirements.

3.1. Wireless Sensor Network for Data Acquisition. The integration of ubiquitous computing in healthcare was first developed by a team of University of Karlsruhe where smart sensors are used together with personalized mobile computing systems [23]. They introduced the concepts of Body Area Network, Personal Area Network and Wide Area Network. The differences between them are provided by the area of monitoring and the communication between devices:

- Body Area Network (BAN) - the monitoring area consist in sensors and devices near the patient body.
- Personal Area Network (PAN) - the monitoring area is the patient's environment.
- Wide Area Network (WAN) - PAN are connected to the central server.

The assisted living systems use various data such as: human body temperature and humidity, EKG signal variations, blood pressure, breathing frequency, resident's activity of the daily living, pulse-oxymeter data, motion data etc. Our system uses a Wireless Body Area Network (WBAN) [20] which integrates intelligent monitoring devices. A WBAN is a set of physiological and / or environmental sensors an their number depends on the end-user application. Using a WBAN we propose a middleware that it is characterized by:

- support for sensor data acquisition and processing;
- support for reasoning, interpretation and decision over the sensor data;

- support for alert system;
- scalability;
- the flexibility and easy configuration in order to support a wide range of protocols and sensor nodes;
- integration in a wide range of software systems (house intelligent systems, vehicle monitoring systems, etc).

All the received data needs to be stored in a database, or to be use in different inference / aggregation processes or to be interpreted. The sensor data interpretation is an annotation, due to a Jess rule that has as left-side the sensor data value and as right-side the corresponding label. The sensor data interpretation corresponds to the first level of sensor data processing described in Joint Directors of Laboratory [25]. The inference means that we obtain new information using interpreted sensor data and applying over them inference rules. For example the human body temperature must to be stored in the database, only if there are some variations related to a temperature higher than 37 Celsius degree. The nurse or the doctor use the received data to make some diagnoses or predictions about the future patient's life, about his medical treatment, or about anything else that is important for him. To middleware gives the sensor data to doctors and / or nurses in a readable format, and to assure this it must to do some interpretation or inference.

The wireless sensor network is composed of sensor nodes, a sink node and the gateway. The WSN takes into consideration three layers (Figure 3.1).

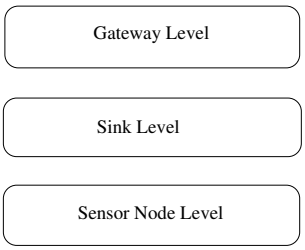

FIG. 3.1. *WSN Middleware Layers*

The *sensor node layer* is used to describe the communication protocol between the simple node and the *sink*. The node layer uses nesC [1] capsule programs which describe this protocol [15, 14]. The simple nodes only send regularly data gathered from the environment. There is no other responsibility for the sensor node that to send data.

The *sink layer* is used to receive data from sensor nodes, to build data-streams for each sensor node and to send them to the gateway.

The *gateway level* is used to collect the received data from all the sensor nodes and to provide some useful function such as filtering and storing data. Our middleware is developed for the gateway level. Our approach uses a simple perspective for the simple sensor node comparing with another WSN context-aware system which gives a lot of responsibilities to the node layer. The proposed system does not load the node layer or sink layer with a lot of functionalities, it uses intensively only the gateway level to process all the gathered data. There are other approaches that use the simple node layer for data-stream aggregation, interpretation, normalization, etc. In our approach all these data-stream operations take place at the gateway level. The context-aware healthcare system must respond in real-time and with a good accuracy. It is important to obtain quickly the vital signs of one patient in order to help him immediately. For this reason many operations that can take place at node level are deployed at the gateway level, and the only responsibility of node layer is to send data with the established frequency, or to receive query messages from the sink node. We adopt this approach because the node layer has many limitations about processing speed, memory, energy etc. Because there are these limitations, it is obvious to move a lot of responsibilities to the gateway level.

The gateway level receives streams and has the responsibilities to provide data which describe the context that can be used in our system. If it is necessary, the gateway level makes some data interpretation or applies some inference rules. For example, if at one instant the received temperature is 37.5, this temperature is translated by the middleware in "sub-feverish". Also if the moving detection sensor sends the value 0.05, this

[1] nesC is an extension of C programming language used to build applications for sensor nodes operating systems, such as TinyOS.

means that the patient in the emergency room is "sleeping" or is "watching TV". The middleware has the responsibility to detect some situations that can be translated in "alert".

Data interpretation means a translation of received streams in a relevant way for the application, or for the human user who does not understand the value of sensor received data. For better sensor data interpretation it is useful to use an ontology that provides support for semantic data annotation [36]. These ontologies are used also in the inference process over the interpreted data-streams. It is a necessary to interpret and inference the sensor data at the middleware level and not to demand special engines built for this purpose, because the system must quickly respond to the environment changes.

3.2. Multi-Agent Architecture. The architectural view of the gateway level is described in Figure 3.2.

The middleware builds agents for each sensor node. Each agent has as main goal the data-stream processing of the corresponding sensor. The use of a multi-agent architecture is motivated by the middleware complexity, adaptability and continuity requirements.

The data-stream is processing to obtain new and useful data, and in many cases is mandatory to do some abstraction. For each level of abstraction we use dedicated agents. We can have on this way an abstraction hierarchy.

A set of sensor data can describe a situation. Seng W. Loke in [26], formalizes and explains how to do reasoning about situations in order to obtain new data or to take correct decision. We can identify new situations continuously acquiring data about a person vital signs such as temperature, blood pressure, pulse frequency. The acquisition process is about the patient short-term, medium-term or long-term behavior. The agent "memory" store data to reasoning on short-term, medium-term or long-term. All these data will be interpreted, evaluated and compressed. The logical predicates of Seng W. Loke will be replaced in our architecture by a set of agents. These agents are used to identify states or state transitions of the patient from which the WBAN collects data. Examples of situations are:

```
(S1)Patient X is sleeping
(S2)Patient X has fallen down
(S3)Patient X is eating
(S4)Patient X has not respond to a call
```

To detect all of these situation there are agents that use some inference rule to obtain new data. The sensor data translation in understanding data is the first abstraction level. Detection of a simple situation, like in the above examples, is the second abstraction level. The simple situation describes a patient state, such as: is sleeping, has falling down. The simple situation can be described using only one inference over the sensor interpreted data. We can continue to do some inference over the last data, as example:

(Patient X has fallen down) AND (Patient X is sleeping) => (Emergency Situation about Patient X)

In the above situation we obtain a new data using another inference. In this way we use interpretations and inferences over the received sensor data in order to provide for the final user understandable and useful data. Identifying situations, the system does its job without human intervention.

The number of agents depends on the received sensor data and to the required abstraction level. The data-flow is changing and depends on the data itself. It is impossible to take in to consideration all the scenarios that can be happen in order to define data-processing workflows. The connections between these agents are not direct or predetermined. So, to respond on this indirect and dynamic environment a *multi-agent architecture* based on blackboard model [10, 30] is the appropriate solution.

The *blackboard* on our architecture is a shared memory that can be used to give an overview for all the agent. The blackboard contains temporary all the received data-streams. It's special goal is to store the incremental view about the patient context. Additionally to sensor received data, this repository contains also the result of data interpretation and aggregation produced by some agent behavior. We have a three-level blackboard. At the first level there are sensor received data. On the second level we find sensor received data interpreted. On the third level we find the inference data result such as: "Patient X is sleeping", "Patient X has not respond to a call", etc. The inference results over the interpreted data are stored on the third level. The blackboard contents is changing dynamically. The blackboard is a temporary buffer with a limited storage capability. The blackboard data that is never used for a long period of time is deleted. For each data-type we have define a time-limit. We use an algorithm to delete data, based on data relevance, insertion time, and the time-limit to be store on the blackboard. The sensor received data are stored on the blackboard for a short time (only a few seconds), and the interpreted or aggregated data are deleted only when there are not used for a long time period.

Our blackboard metrics are: the insertion-time and the relevance. For each data we store the insertion time on the blackboard contents, as the time when we received it from the sensors or from middleware agents. We define four relevance degrees:

- 0 – relevance for the sensor received data;
- 1 – relevance for the interpreted sensor data;
- 2 – relevance for the data obtain by using inference rules over the interpreted sensor data;
- 3 – relevance for the alert signals;

The less relevant data are the sensor received data-stream. The more abstract and useful is the most relevant data. The highest relevance is allocated to alert data. Also the blackboard keeps an index for all the data. When we insert a new data on the blackboard, we calculate an index value for it. This index is a unique number, and it is assigned for each new data entry. The blackboard uses a queue where the most relevant data are stored for a long time, and the non-relevant data are deleted from the queue.

We have defined rules that allow to build new agents. We have **Simple Sensor Receiver Builders** used to build agents only for sensor stream processing. Also, we have **Logic Sensor Receiver Builders** used to build agents that allows data aggregation / inference or interpretation. De facto, these builders are the knowledge sources for the blackboard. The result of each builder activations is a new agent. The builders are a set of rules that are activated when on the blackboard we find some special data. A special data corresponds to the builder rule left-side patterns. As example we have two builder: AmbientTemperatureBuilderAgent and HumanBodyTemperatureBuilderAgent. For the first builder a special data, is a number between -5 and 35, without decimals, and for the second is a value between 35 and 42, with one or two decimals. In this way the blackboard can be viewed as a reactive database. For example when we find on the blackboard temperature sensor data in the range 36-40 Celsius degree, the sensor receiver builder will define a Simple Sensor Agent for Biomedical Temperature. Agent's definitions are available on the middleware. An agent definition contains the situation when it must be build and it's core functionalities, scenarios and a list of ontologies that can be load. Using the matching mechanism over the blackboard data, the JESS rule engine is using to activate the building process. These builders or knowledge sources are developed using the direct communication with the final user or by learning. These builders contains some expertise about some types of data. These expertise contain patterns that allow the Jess rule left-side matching process. For the temperature data we can have a builder / knowledge source, for the patient state we can have another builder / knowledge source and in this way we can be sure that the middleware will be populated with a lot of agents that will process entirely all the data-stream from the backboard. Between the builder and the supervisor there is a publish-subscribe mechanism that allows the communication between them. The builders publish the models for the future agents on the blackboard and send a message to the supervisor to announce that.

The supervisor makes runtime decisions about the agents on the middleware. The supervisor has two main functionalities: first it must create agents using the builder proposals, and also it must to manage the correspondence between blackboard data and the middleware agents. Let's consider the situation when for the same data-stream we have in the middleware two or maybe three agents. The supervisor will solve also the conflicts that can appear (related to the blackboard access, agents with similar competencies etc). The supervisor will take decisions about the agent that will survive and about the agents that is redundant and must be killed. Also when a data-stream is broken the supervisor will kill the corresponding agent. The supervisor decides also what agent to create when the builder makes multiple proposals. The supervisor is shared to all the builders, and in this way we assure the communication between the builders and supervisor. The supervisor subscribes to all the builders, and the builders send messages to them. Using these messages the supervisor is informed that the builders have doing some proposal about the future middleware agents. Each builder proposal it is stored on the blackboard, and the supervisor read all these proposals and decides which agent will be built. This happens when the builder has time for storing its proposal on the blackboard. There are situations when is no time for proposals storing and supervisor decision. For the current cost sensor data is useful to issue proposals and to choose from these proposals, but for the blood pressure sensor data this is not a good practice.

3.3. Simple Sensor Agent, Logic Sensor Agent, Virtual Sensor Agent. At the gateway level, we have data aggregation or the inference process over the interpreted data. The main data processing workflow is:

sensor data acquisition → interpreting data → aggregates information.

For example let's consider the situation where the moving detection sensor sends the value 0.05, the time is 23:45, the heart rate is 63 $beats \cdot min^{-1}$. These information are interpreted as it follows: No Movement, Night, LowHeartActivity. The aggregation result about the interpreted data is: the patient is sleeping [39]. The proposed architecture gives to the node level the responsibility to send data, to the sink level the capability to collect data, and to the gateway level the processing data operations, data-stream interpretation and aggregation.

The middleware uses:

- A Dispatcher Sensor Stream Agent (DSSA) that has as main responsibility to receive the signal from the physical sensors and to create a temporary stream warehouse for the SSA.
- Builder Agents that do proposals for the future middleware agent. They write their proposals on the blackboard.
- A Knowledge Discovery Agent that explore another external systems or databases in order to obtain new and useful data. All the data are stored on the blackboard.
- Simple Sensor Agent that are used for the sensor data-stream processing. The processing results are stored on the blackboard.
- Virtual Sensor that indicates some useful time-series data that can be produced by the middleware.
- Logic Sensor Agent that are used to obtain new knowledge over the data from the blackboard: sensor-data, data from another external sources, data gathered from the SSA.

All the interactions between the middleware agents are mediated by the blackboard. Each agent stores its output on to the blackboard, and another agent is activated to process this output if the situation requires this. There are no direct interactions between two SSA or two LSA. There is a data-flow that use a lot of agents, but it is mediated by the blackboard.

Using a lot of software agents that process each sensor data-stream we solve the complexity issue. The WSN uses a lot of sensors. We can have for each sensor category one or more agent. There are situations where a sensor family measure the same fact: in a room there are a lot of sensor temperature used to detect the average temperature. In another situation we have the same kind of sensor but all of these sensors are used to measure different facts. Let's consider in this situation current cost sensor used for each house's appliances. For each sensor data-stream our middleware use **Simple Sensor Agent** (SSA) . The SSA has as responsibility the sensor data-stream processing. The main SSA responsibility is the signal acquisition from the corresponding sensor or sensor family. The SSA is able to understand the received data-stream and to detect all the events in this time-series. The agent's behavior can be changed if special events have been detected in the data-stream. As example let's consider the situation when in the data-stream a strange value was discovered ; in this situation the SSA must to detect if this strange value is a noise or not, and in this way it changes his classical behavior. The main goal of each SSA is to process the received data-stream in order to obtain useful new data. So, the agent's goal does not require a strong proactivity. It is important for the SSA to be more reactive than to be proactive.

There are situations where for each sensor the middleware build a SSA and situations where we build one SSA for a sensor family. When we have a lot of temperature sensors (a sensor family) that measure the same human body, like in the above example, the middleware must build one SSA. When we use sensors for independent phenomena the middleware built different SSA.

The data-stream interpretation or aggregation is due to **Logic Sensors Agent**(LSA). The LSA has the same functionality as SSA, the difference between them is that the LSA is working with the output of SSA or the output of another LSA. In section 3.1 we explained how we can create some abstractions hierarchy, that are translated in one or more LSA hierarchies.

Our architecture also uses **Virtual Sensor Agents** (VSA). Let's consider a first example of a VSA for that encapsulate the time. When we detect that a patient is sleeping we use a detection sensor, a heart-rate sensor and a VSA for the time.

3.4. Dispatcher Sensor Stream Agent (DSSA). An important agent of this architecture is the *Dispatcher Sensor Stream Agent*. This agent connects with the WSN gateway. Our gateway receives all the sensor signals and send them to this agent. The "Dispatcher Sensor Stream Agent" scans all the detected streams, selects only the representative data-stream, and store these data for a very short period of time - 30 seconds as example. The representative sensors are the sensor which will be use by the AAL system, for which the middleware has an acquisition goal. The received data-streams from the representative sensors, are called representative data-streams.

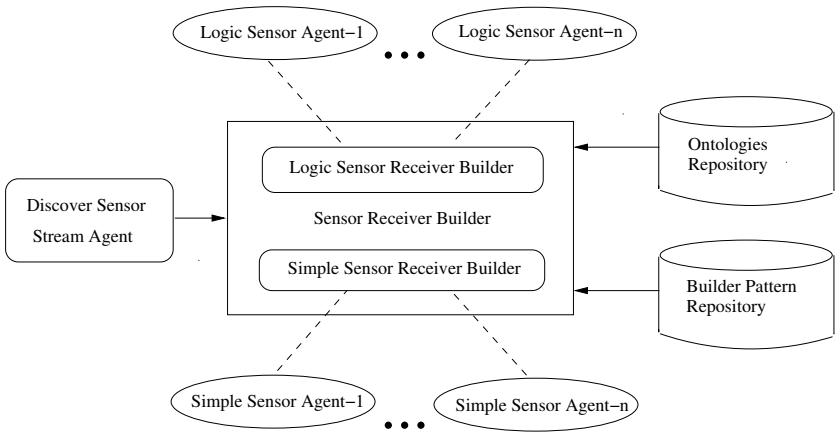

FIG. 3.2. *Middleware Components*

The assisted living systems use a limited sensor data-stream. When in a room we have a lot sensors that send data packets, the middleware receives only the sensor data-stream for which is an acquisition goal. Suppose that in a room we have sensor for ambient temperature, moving detection, luminosity, optical meter for gas consume, electricity etc and also a body sensor network - accelerometer, human body temperature, humidity, etc. The gateway will receive all the signals from all the sensors, and all these data-stream will be stored on the DSSA for a short time period. The AAL system considers as representative sensor, on this example, only the ambient temperature, moving detection, accelerometer, human body temperature and humidity sensor. The DSSA is a temporary buffer that it is used by all the agents. From this buffer the simple agents "extract" their data-stream. The DSSA represents the first blackboard level, because inside it, is a stream warehouse for all the SSA.

There is a ZigBee communication between the sensor nodes. The ZigBee communication is developed using XBee modules for each sensor node. In this way each sensor node has a unique identifier, the Personal Area Network - ID (PAN-ID). By default, for each device the PAN-ID is 3332 and the sensor network administrator must to change this ID below to integrate the sensor nodes in the network and to start the middleware. The WSN main feature is the heterogeneity, because there are sensor nodes for temperature, heart rate, moving detection etc. The WSN heterogeneity is due to sensor nodes physical features and communication protocol. We use a SocketProxyAgent for the communication with sensor nodes and also JESS to initiate this communication. The SocketProxyAgent can be used for maximum 50 parallel connections and also has limitations about the communication protocol. For solving this problem we can also use JESS, to create a socket connection with the sensor network, and to read the received packet. Also for the middleware scalability we can use many SocketProxyAgents. This solution is more complicated than the Jess solution, because it implies the management of all the agents and also we can have the risk of a broken connection and this can change the agent behavior.

Our middleware will process only the representative data-stream: in our case only the data from the accelerometer, human body temperature and humidity, ambient temperature and moving detection sensor. The XML file SensorDataModel describes the data template for the representative sensor. The data template for the representative sensor describes:

- how many fields has a received sensor-data packet;
- each field of this packet: his position on the packet, beginning from left to right, his name, his minimal / maximal value and the unit used to measure this value.

The middleware is able to be configured to select and to operate with some data stream. For this it is necessary to rigorously define the representative sensor data-streams. The XML file SensorDataModel describes the acceptable middleware package data formats. If a data package is not corrupted and this package data-type is not in the SensorDataModel.xml the middleware does not accept this package. There is the possibility to add new package data-type, but this operation is allowed only with human user assistance. For example, the temperature sensor data for a human patient is described as follow:

```
<temperatureSensorModel1>
  <fieldsNumber>3</fieldNumber>
  <field>
     <position>1</position>
     <name>Time</name>
     <minValue>0</minValue>
     <maxValue>89400</maxValue>
     <unit>S</unit>
  </field>
  <field>
     <position>2</position>
     <name>Temperature</name>
     <minValue>0</minValue>
     <maxValue>43</maxValue>
     <unit>C</unit>
  </field>
  <field>
      <position>3</position>
      <name>ID</name>
      <minValue>0</minValue>
      <maxValue>100<maxValue>
      <unit/>
   </field>
</temperatureSensorModel1>
```

For each sensor data-stream a sensor receiver agent is built with the responsibility to manipulate this stream.

3.5. Knowledge Discovery Agent. This agent is used in our architecture only to extract data from different external databases or healthcare systems and to process this data. There are a lot of situations where there is a strong requirement to have a general view about the problem and some information is missing from the local system. In these situations the Knowledge Discovery Agent (KDA) is used to "travel" along different external data-storage repositories and to extract some useful data. On the context of AAL systems, data is distributed over multiple organizations such as hospitals, departmental healthcare-insurance office, etc.

The knowledge discovery process is composed of several phases such as: business understanding, data understanding, data preparation, modeling, evaluation and deployment [9]. For each phase of the discovery process we define an operator that is used to achieve the step goal.

The first task of this agent is to define correctly data about which it is interested. The KDA will analyze data from external sources and will compare this data with its descriptions. When we have a patient that changes his hospital in a period of one week, it is required to obtain some data stream about his cardiac frequency from the last three days. The KDA will inspect a lot of external databases that contains data stream about this patient cardiac frequency, but will select only the data from the last three days. In order to complete his task the KDA will collect data about this patient, and our system, locally, will select the relevant information. The agent query will be more general, to obtain in this way a lot of data about this patient. It is more efficient to collect all data in one agent "travel" than to do more "travels" for each query. So, we can use one or more KDA agents to obtain relevant data about the patient: one agent will travel to collect data and others will process data locally. The file SensorDataModel describes data received from a local sensor, and in the same way another file ExternalDataModel describes data that must be extracted from another external databases or systems. This file describes the query over another external databases and data quality. The extracted data can satisfy all the requirements from the ExternalDataModel files, or partially. A metric will be used to say exactly to how many requirements the extracted data corresponds. Of course, the KDA will use a fuzzy approach to select the relevant data. So the second KDA agent's task is to define the extracted data quality, and another task is to select the relevant data.

The last task KDA agent's is data preprocessing. The agent must prepare data to be used by another middleware agents. This means to translate the extracted data in a form that allows the processing at the middleware level.

This agent can use some transducer agents that allow the communication between the KDA and the external database [16]. The requests must be translated from the ExternalDataModel to ACL (Agent Communication Language) or to native requests over the external data sources.

3.6. Sensor Agent. The Figure 3.4 describes the sensor agent internal architecture. This design integrates three layers: *behavioral level, the core level and the semantic layer*. The core layer is mandatory and the behavioral and semantic layer are optional. The sensor agent behavior consists of a lot of operators that can be used to manipulate the data stream. The sensor operators implement a generic interface. In this way it is possible to define new operators, using these interfaces. For example we can have filter operators, matched-filter operators, persistence operators, transformation operators, aggregation operators etc. The semantic layer loads an ontology in order to "understand" the received sensor data-stream. The understanding process of sensor data is "de facto" due to the interpretation or inference over the received information.

The agent core layer contains all the mechanisms that are used to load the ontology, to define the agent's behavior and to be context-aware. The agents use a pull mechanism to obtain the corresponding sensor data-stream from the blackboard. This mechanism is developed on the core layer. The agent communication with the blackboard is defined on the core layer. This layer contains also the primitives for the initialization and destruction of each agent. In order to ensure the context-aware agent feature is important to store a knowledge base that allows to be reactive at the environment changes. A set of rules are stored on the core layer. The core layer has two main responsibilities: scenario recommendation and its execution. The agent behavior is defined by a scenario, that is an operators workflow. There are a lot of workflows definitions available in a special repository. The workflow is selected using an evaluation formula [24, 34] build based on the ISO / IEC 9126 standard [1].

Another important responsibility is the communication between the agent and the physical sensor or sensors. We don't use the Plug-and-Play mechanism, we read all the received sensor data-stream and we select only the representative data-stream in our middleware. For this reason the agent must communicate with the physical sensor. The agent calculate and allocates the PAN-ID for the sensor / sensors, and it can initiate some special query over the sensor / sensors.

3.6.1. Operators over data-streams. The sensor agent behavior is implemented as a directed graph of operators. Thus, we choose a directed graph as container for the operators because the sensor's behavior must respond to different situations. The problem of sensor data-stream persistence is solved using the persistence operators, which use a reflexive mechanism. To support these operators structure it is also necessary to define some decisional operators.

Our architecture defines three categories of sensor data stream operators: data-stream segment operators, time-set data operators, singular record operators. The proposed operators architecture respects the requirements presented in [35]. The data-stream segment operators operate over a set of received data in a short or long period of time: five minutes, one hour, one day, etc. A temperature-sensor data stream segment can be the following:

```
23:49:40. 37,1
23:49:50. 37,1
23:50:00. 37,1
23:50:10. 37,1
23:50:20. 37,1
23:50:30. 37,1
23:50:40. 37,0
```

In this category we have the following operators:

- data-stream completing operator. This operator replaces the missed value from the data-stream. As example, if in the last data-stream segment the record (23:50:30. 37,1) will be missing, the middleware must complete the data stream. The inserted value will be the average between the record before an the after-record.
- data-stream de-completing operator. This operator deletes all the irrelevant data from the data-stream. In the above example we have the same temperature value from 23:49:40 to 23:50:30. The de-completing operator will delete all the records from this interval or will be reduce the number of records with a specified percent.

- event detecting operator. This operator is used to detect a relevant event from the statistical or semantic point of view. The definition of these operators implies in most cases a pattern recognition engine embedded in the agent.
- inflexion point detection on data-stream operators. This operator has the same behavior like "data-stream de-completing" operator with the difference that it is used to detect if there is a dependency between the time and the sensor value. If there is a dependency, this operator deletes the value that can be calculated and store only the inflexion points (the points where these dependencies are changed).
- statistical operators such as: min, max, average, etc.
- redundant data deleting operators. Redundant data is defined by the user. In some situations redundant data like in the example of "data-stream de-completing operator" or "inflexion-point detection on data-stream operators".
- noise elimination operators. This operator delete some abnormal values. In the above data-stream segment an abnormal value is -37.1 at the moment 23:50:20.
- filtering-operator. This operator identifies some special values in the sensor data-stream, or some pattern in this data-stream. The filter is specified by the user.
- data-stream fusion operators. This operator is used when the segment contains a lot of records and we want to reduce this number. Some other possibilities are to use the de-completing or inflexion point detection operator. This operator will use some statistical method in order to reduce the records number: it deletes a lot of stream-values and introduces new values calculated using some operations like general average over stream-values.
- historical data stream interpretation operators. These operators are used to do some interpretation for the data-stream using data from the "ancient" time.
- data-stream normalization operators.
- data validation. These operators validates the data-stream segment from the semantic point of view.
- stream-to-cube operator, used to produce a cube using the received facts. It is triggered by time or tuple-based events.
- distinct element detection operator over the data-stream segment.
- distinct elements counter operator over the data-stream segment.
- distinct elements frequency operator over the data-stream segment.

Time-set data operators operate over a set of data received at the sensor and gateway level at the same time, that can be manipulated by the same middleware agent. An example for this is the situation when for a patient we consider at the same time more EKG sensors. For this category we can consider the following operators:

- data-interpretation operators.
- aggregation operators.
- fusion operators.
- filtering operators.
- database - storage operators.
- statistical operators such as: min, max, average, etc.

The singular record operators operates over a single record from the data-stream. In this category there are operators used for data-interpretation, database-storage, event-detection and the most important is the alarming operators that detect some special situation and send an alarm message.

Our architecture is open to define new operators for sensor-data stream. In this way, we have define an interface for each category of the above operators. Each new operator must implement an interface. Also we have define inverse operators such as:

- inverse storage operators which defines a data-stream using the stored information;
- inverse interpretation operators that builds the data-stream doing an inverse interpretation.

The sensor behavior is "de facto" an operators workflow. Due to the agent reactivity, the behavior is implemented with a context-sensitive operators workflow.

3.6.2. Ontologies. The interpretation of sensor data-stream is done using some special operators designed for this goal. For a good interpretation or inference, the semantic layer will load an adequate ontology.

The details about the context can be manipulated using different model such as: key-value model, the logic model, the object oriented model, etc. The paper [36] presents a comparative study about the models that can be used for context representation. We chose to use the ontological paradigm to define the context because

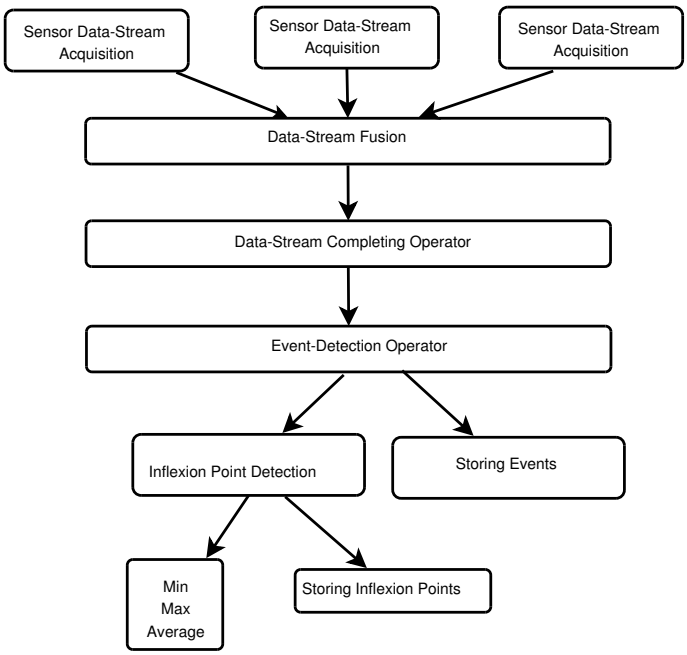

FIG. 3.3. *Sensor Data Stream Operators Workflow*

this model gives support for reasoning, validation and distributed composition. Our system uses different data repositories from different location. On each location we can have different interpretation for the relevant data. So, when we import data from a repository we must to import the ontology and / or the meta-model that describes this data. This means that the ontology model should respond to the distributed composition requirement. Our data must to be validated in a specified context. There is a difference between how we validate the cardiac frequency for a patient with several cardiac attacks or for a normal patient. For each patient we have a different context, and this context is implemented in our middleware using ontologies.

The agent's semantic layer will load the adequate ontology in order to support the context-aware approach. This application will be use in different contextual situation, as example, in a hospital emergency rooms and also in a house for the surveillance of patients. The interpretation and data aggregation must take in consideration the patient's context, his medical background etc. To support these different contextual environments the definition of context based on ontologies is the better choice. When the assistive system "run" for an hospital emergency room will load for all the agents the ontology that describes an emergency room [18]. In another situation when the situation is about elderly people the semantic layer will load the ontology that describes elderly people. The patient's sex also will customize the used ontology. The interpretation of some vital signs depends by patient age or sex. The core of the middleware offers facilities to transfer the contextual information to another sensor agent using the suitable ontology. In this way the middleware can be easy adaptable to different environments.

The context changes over time, so we need to define the past context, using adequate ontologies [22]. Data interpretation depends by the acquisition time. If a person has falling down and at this moment he is in a fix point, the moving detection sensor data is for example 0.05. In a normal situation at the same time of the day the moving detection sensor data has higher value like 0.95. The middleware will load the ontology that describes people that are falling down, an in this way we can obtain a good context overview.

Intelligent ambient systems manipulates large amounts of data, with a high degree of heterogeneity. The ontologies give support to the data heterogeneity and also allow the interoperability. The context information must be commonly understood by all the agent [31]. Another important requirement is related to a more general character of data. Ontologies give support to define concepts hierarchies. The middleware architecture use intensively the Dependency Inverse Principle [27], so the agent are working with more abstract "objects" that are defined using ontologies.

The middleware uses an ontology repository, that is used to describe the context in different situations. The middleware selects the adequate ontology from the above repository, that depends on the following features:

- patient demographic characteristics like: sex, age;
- patient medical background: past diagnostics, medical events in his life.
- patient monitoring context: emergency room in a hospital, patient house, etc.

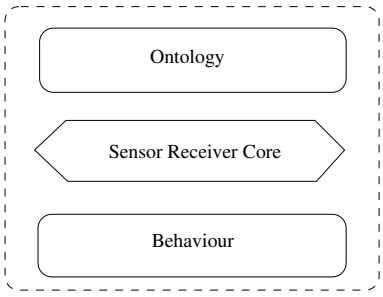

FIG. 3.4. *Inside the Sensor Receiver*

Our ontologies are build around the concept of event. Our middleware has as main goal the event detection on the sensor received data-stream. The main concepts of our ontologies are: patient, event, activity. There are three main event categories: about the subject of our system - the patient- and about the data-stream and about the environment description. The ontologies about the patient contains concepts and predicates that describes exactly the patient state, his actions, his medical background, etc. The data-stream ontology describes the events that can be appear in the received data from the sensor: sensor stops, sensor interruption, sensor changeovers, etc. The environment ontology describes all the events related to environment: the patient environment (EKG, pulse, oxymeter, moving data etc), the room environment (opening a windows / door, cleaning the room, etc), the sensor environment (sensor start event, sensor interruption event, etc).

More exactly our ontology is focused on the event that allow transitions between different states. We chose the event approach to describes the ontologies because our middleware is a real-time software that allows rapid intervention of some specialists (doctors, nurse etc). Our ontologies are more dedicated to the development of predicates that allow the event recognition, than to the concepts definition. Previous ontologies for assisted living system have been proposed in [22, 8, 33]. Comparing with SOPRANO project ontology we focus on the event, not to the state, as central ontology concept. Our approach uses the ontology to a better context understanding, because the agents load the adequate ontology, and we are not working over the ontology like in SOPRANO. On the above project, the ontology is used as a contract between the system components in order to give semantic coherence. Our middleware loads the adequate ontology and shares to all the agents. Our ontology system models the sensor events that allow the acquisition, on the context of an AmI system.

4. Related Work. The gap between the assisted living system and the wireless sensor network represents the necessity to develop sensor middleware. The middleware concept is frequently used in distributed systems. The idea that a sensor network is a distributed system gives reason that a middleware is a solution for this gap. Middleware is an approach to satisfy all the requirements presented in the second section in order to provides services to assisted living system. This section examine the existing middleware for WSNs.

MiLAN [19] uses as an input the following information about: variables of interest for the software application, the QoS for each variable and the QoS level. According to these inputs and the sensor collected data, the middleware provides as output a stream that will be used by the application. To discover and to obtain information about the sensors MiLAN uses a service discovery protocol. MiLAN runs different types of application and need some adaptation to provide services for them.

A related project, IrisNet (Internet - Scale Resource - Intensive Sensor Network Services) middleware use an external storage approach [17, 29]. It uses XML to represent sensor - produced data. These middlewares use software agents to collect and to organize data. There are two kind of agents used in IRISNET: sensing agent and organizing agent. The sensing agents are used to access sensors, and the organizing agent implements a distributed database of services for the collected data. IrisNet uses an adaptive data placement algorithm provided by organizing agent.

AMF (Adaptive Middleware Framework) [40] is a sensor middleware that enables the reduction of the energy consumed in the process of sensor information collection. It uses the "predictability of sensor readings" to improve the efficiency. It provides precision and prediction-based adaptation .

5. Conclusions and future works. This paper introduces a new multi-agent approach for a context-aware sensor data middleware used in healthcare and assisted living systems. Our approach uses the ontology definition of context and introduces some special components to manipulate the sensor data-streams. A original aspect is related to the use of logical sensor agent in order to obtain new knowledge. Comparing with similar approaches [22] our middleware uses ontologies for a contextual understanding. This architecture based on the multi-agent model is scalable, extensible and provides openness to define new operation and adequate workflows for stream manipulation or to define a new way to understand the context.

Our middleware provides support for reasoning at different levels of abstraction over the received data. The data repository store the sensor received data and also high-level information provided by the logic sensor receivers. The main function of the related middleware is to collect the sensor data, but our solution gives support to develop intelligent assistive systems. An ambient intelligent system can be described with three keyword: sensing, thinking, reacting. The described middleware provides services for sensing and support to thinking about the context. Our approach introduces the idea of reasoning over the sensor data, and in this way the assistive system core has as responsibility to take the correct decision and to execute these decisions. This feature is provided by the use of logic sensor receivers and ontologies that describe different situation.

We will take in consideration to describe different context using adequate ontologies. First, this means building ontologies and also giving a mechanism to select the better ontology that describes an environment or a situation. A lot of interest will be focused to the development of a framework that allows data-stream operators, ambient assisted living ontologies, builder pattern and workflow definition. A prototype will be developed on JADE [7].

We will develop our middleware in order to be used not only for assisted living systems. We try to develop a generic middleware for data-stream that can be used in other ambient intelligent systems: home-control systems, traffic control systems, etc.

REFERENCES

[1] ***, *ISO/IEC 9126-1:2001 - Software Engineering. Product Quality*, ISO, Geneva, Switzerland, Jan. 2001.

[2] ———, *ISO/IEEE 11073-30300:2004 - Infrared wireless*, Health informatics – Point-of-care medical device communication – Part 30300: Transport profile, ISO, Geneva, Switzerland, Jan. 2004.

[3] ———, *ISO 12967-1:2009 - Enterprise viewpoint*, Health informatics – Service architecture, ISO, Geneva, Switzerland, Jan. 2009.

[4] ———, *ISO 13606-3:2009 - Part 3: Reference archetypes and term lists*, Health informatics – Electronic health record communication, ISO, Geneva, Switzerland, Jan. 2009.

[5] J. C. Augusto, H. Nakashima, and H. Aghajan, *Ambient intelligence and smart environments: A state of the art*, in Handbook of Ambient Intelligence and Smart Environments, H. Nakashima, H. Aghajan, and J. C. Augusto, eds., Springer, New York, 2010, pp. 3–31.

[6] J. E. Bardram, *Applications of context-aware computing in hospital work: examples and design principles*, in SAC '04: Proceedings of the 2004 ACM symposium on Applied computing, New York, NY, USA, 2004, ACM, pp. 1574–1579.

[7] F. L. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with JADE*, Wiley, 2007.

[8] Y. Cao, L. Tao, and G. Xu, *An event-driven context model in elderly health monitoring*, Ubiquitous, Autonomic and Trusted Computing, Symposia and Workshops on, 0 (2009), pp. 120–124.

[9] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, and R. Wirth, *Crisp-dm 1.0 step-by-step data mining guide*, tech. rep., The CRISP-DM consortium, August 2000.

[10] D. D. Corkill, *Collaborating Software: Blackboard and Multi-Agent Systems & the Future*, in Proceedings of the International Lisp Conference, New York, New York, October 2003.

[11] R. Costa, P. Novais, L. Lima, D. Carneiro, D. Samico, J. Oliveira, J. Machado, and J. Neves, *Virtualecare: Intelligent assisted living*, in Electronic Healthcare, vol. 0001 of Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Springer, 2009, pp. 138–144.

[12] K. Fishkin and M. Wang, *A flexible, low-overhead ubiquitous system for medication monitoring*, Tech. Rep. IRS-TR-03-011, In: Intel Research, 2003.

[13] C. Floerkemeier and F. Siegemund, *Improving the effectiveness of medical treatment with pervasive computing technologies*, 2003.

[14] D. Gay, P. Levis, D. Culler, and E. Brewer, *nesc 1.1 language reference manual*, 2003.

[15] D. Gay, M. Welsh, P. Levis, E. Brewer, R. von Behren, and D. Culler, *The nesc language: A holistic approach to networked embedded systems*, in Proceedings of Programming Language Design and Implementation (PLDI), year = 2003, pages = 1–11.

[16] M. R. Genesereth and S. P. Ketchpel, *Software agents*, Commun. ACM, 37 (1994), pp. 48–53; 147.

[17] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan, *Irisnet: An architecture for a world-wide sensor web*, IEEE Pervasive Computing, 2 (2003).

[18] T. Gu, H. K. Pung, and D. Q. Zhang, *A service oriented middleware for building context-aware services*, Journal of Network and Computer Applications, 28 (2005), pp. 1 – 18.

[19] W. B. Heinzelman, A. L. Murphy, H. S. Carvalho, and M. A. Perillo, *Middleware to support sensor network applications*, Network, IEEE, 18 (2004), pp. 6–14.
[20] E. Jovanov, A. Milenkovic, C. Otto, and P. de Groen, *A wireless body area network of intelligent motion sensors for computer assisted physical rehabilitation*, Journal of NeuroEngineering and Rehabilitation, 2 (2005), p. 6.
[21] J. Kjeldskov and M. B. Skov, *Supporting work activities in healthcare by mobile electronic patient records*, in Computer-Human Interaction, Springler, 2005, pp. 191–200.
[22] M. Klein, A. Schmidt, and R. Lauer, *Ontology-centred design of an ambient middleware for assisted living: The case of soprano*, in Towards Ambient Intelligence: Methods for Cooperating Ensembles in Ubiquitous Environments (AIMCU), 30th Annual German Conference on Artificial Intelligence (KI 2007), Osnabrck, September 10, 2007, T. Kirste, B. Knig-Ries, and R. Salomon, eds., 2007.
[23] C. Kunze, U. Grossmann, W. Stork, and K. Muller-Glasser, *Application of ubiquitous computing in personal health monitoring systems*, Biomed Tech(Berlin), 47 (2002).
[24] Y. Lei and M. P. Singh, *A comparison of workflow metamodels*, 1997.
[25] J. Llinas, C. Bowman, G. Rogova, A. Steinberg, E. Waltz, and F. White, *Revisiting the jdl data fusion model ii*, in In P. Svensson and J. Schubert (Eds.), Proceedings of the Seventh International Conference on Information Fusion (FUSION 2004, 2004, pp. 1218–1230.
[26] S. W. Loke, *Representing and reasoning with situations for context-aware pervasive computing: a logic programming perspective*, Knowl. Eng. Rev., 19 (2005), pp. 213–233.
[27] R. C. Martin, *The dependency inversion principle*, C++ Report, 8 (1996).
[28] S. Mitchell, M. D. Spiteri, J. Bates, and G. Coulouris, *Context-aware multimedia computing in the intelligent hospital*, in 9th ACM SIGOPS European Workshop, ACM Press, 2000, pp. 13–18.
[29] S. Nath, P. B. Gibbons, and S. Seshan, *Adaptive data placement for wide-area sensing services*, in FAST'05: Proceedings of the 4th conference on USENIX Conference on File and Storage Technologies, 2005, pp. 4–4.
[30] P. Nii, *The blackboard model of problem solving*, AI Mag., 7 (1986), pp. 38–53.
[31] D. Preuveneers, J. V. den Bergh, D. Wagelaar, A. Georges, P. Rigole, T. Clerckx, Y. Berbers, K. Coninx, V. Jonckers, and K. D. Bosschere, *Towards an extensible context ontology for ambient intelligence.*, in EUSAI, P. Markopoulos, B. Eggen, E. H. L. Aarts, and J. L. Crowley, eds., vol. 3295 of Lecture Notes in Computer Science, Springer, 2004, pp. 148–159.
[32] S. Shenker, S. Ratnasamy, B. Karp, R. Govindan, and D. Estrin, *Data-centric storage in sensornets*, SIGCOMM Comput. Commun. Rev., 33 (2003), pp. 137–142.
[33] A. Sixsmith, S. Meuller, F. Lull, M. Klein, I. Bierhoff, S. Delaney, and R. Savage, *Soprano — an ambient assisted living system for supporting older people at home*, in ICOST '09: Proceedings of the 7th International Conference on Smart Homes and Health Telematics, Berlin, Heidelberg, 2009, Springer-Verlag, pp. 233–236.
[34] K. Stoilova and T. Stoilov, *Comparison of workflow software products*, in International Conference on Computer Systems and Technologies - CompSysTech, 2006.
[35] M. Stonebraker, U. Çetintemel, and S. Zdonik, *The 8 requirements of real-time stream processing*, SIGMOD Rec., 34 (2005), pp. 42–47.
[36] T. Strang and C. Linnhoff-Popien, *A context modeling survey*, in Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England, 2004.
[37] S. Tilak, N. B. Abu-Ghazaleh, and W. Heinzelman, *Collaborative storage management in sensor networks*, International Journal of Ad Hoc and Ubiquitos Computing, 1 (2005), pp. 47–58.
[38] M. Vallee, F. Ramparany, and L. Vercouter, *A multi-agent system for dynamic service composition in ambient intelligence environments*, in The 3rd International Conference on Pervasive Computing (PERVASIVE), 2005.
[39] M. R. Waldeck and M. I. Lambert, *Heart rate during sleep : Implications for monitoring training status*, Journal of Sport Science and Medicine, 2 (2003), pp. 133–138.
[40] X. Yu, K. Niyogi, S. Mehrotra, and N. Venkatasubramanian, *Adaptive middleware for distributed sensor environments*, IEEE Distributed Systems Online, 4 (2003).

Edited by: Adina Magda Florea, Costin Bădică
Received: February 29, 210
Accepted: March 31, 2010