International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 3 (2011) pp. 862-869
© MIR Labs, www.mirlabs.net/ijcisim/index.html

Towards a Flexible Context-aware Pervasive Alert Generation System

Alessandra Esposito , Luciano Tarricone and Marco Zappatore

Department of Engineering for Innovation, University of Salento,
via per Monteroni, 73100 Lecce, Italia
{alessandra.esposito; luciano.tarricone; marcozappatore}@unisalento.it

***Abstract*: This paper presents a proposal for a context-aware framework for alert generation. The framework is based on a general purpose architecture integrating three core technologies: ontology representation, multi-agent paradigm and rule-based logic. The system is very efficient and versatile and its customization to new scenarios requires a very reduced effort, substantially limited to the update/extension of the ontology codification. The effectiveness of the system is demonstrated by reporting both customization effort and performance results obtained from validation in two different real-life applications in the environment monitoring context as well as in healthcare scenarios.**

Keywords: Multi-agent system, context-aware monitoring, pervasive computing, alert generation system, ontology, rule-based system**.**

I. Introduction

The increasing availability of network connection and the progress in information technology and in hardware miniaturization, are determining new computing scenarios, where software applications are able to "configure themselves" based on information coming from heterogeneous sources (sensors, RFID, GPS, user input, etc.) which form the so called "context". Such computing scenarios find application in a large number of real-life domains, such as monitoring, target localization, auto-diagnostics. Nowadays many different monitoring systems are available: in the health-care domain they perform many tasks such as therapy and treatment planning, diagnostic support, prognosis analysis (see [1] for an up-to-date review). Ubiquitous context-aware systems find also usage in diverse application domains concerning the environment [2], such as water and air pollution monitoring, climate changing tracking, indoor monitoring, etc.

Such a high number of applicative domains casts a fundamental need: the availability of a framework easily configurable and adaptable to different real-life scenarios. For this reason, we designed and implemented a pervasive system for context-oriented monitoring purposes which is based on a flexible and versatile framework. The system is based on the integration between three core technologies: ontology representation, multi-agent paradigm and rule-based logic. The customization of our system to a specific scenario just requires the extension of the available ontology. This is demonstrated in the paper by providing an example of application to two concrete situations: the former in the health-care domain, the latter in the environment monitoring context. Moreover, numerical results are provided as a validation of the system amenability to support real-life situations.

II. Related Work

Our framework is organised according to a general purpose architecture, centred around an ontological context representation.

The amenability of ontologies for building context-aware pervasive computing systems has been already demonstrated in other works. Some of them, such as [3] and [4], both built around OWL and rule-based inference for ubiquitous healthcare, are dedicated to a specific application field, thus being not flexible, and requiring a huge effort to be converted to other fields of application.

Other works, instead, present context-aware systems which are suited to be reconfigured and adapted to different practical situations. Wang et al. [5] propose Semantic Space, a pervasive computing infrastructure based on Semantic Web technologies. Gu et al. [6] propose SOCAM, an ontology-based middleware for context-aware services. Chen et al. [7] describe COBRA, an agent-based framework for intelligent meeting rooms. These systems propose software infrastructures which greatly help the developer in implementing context-oriented frameworks by providing useful APIs, context wrappers, middleware.

Our work differs from previous works in several respects. Rather than implementing a software infrastructure, we developed a reusable framework, able to auto-configure automatically to specific application scenarios. The unique operation requested is the ontology extension. This eliminates the intervention of software developers in system customization. This was made possible by restricting the area of interest to alert monitoring. Moreover, previous works often lack a well documented characterization of customization effort and behavior.

We, instead, provide a twofold evaluation. First, we build up real-world applications. Then, we measure the resulting system's performance.

III. Alert Generation Systems: a General Schema

In order to cover most alert generation situations with a reduced customization effort, we defined a very essential schema (Figure 1) able to describe a generic alert generation system. As shown in figure, on one side we have raw sensed data, on the other side there are the monitoring entities, who are interested in being notified when sensed data have some out-of-range behavior.

Raw data are provided by physical sources which monitor parameters. Several context sources (i.e. parameters) can be associated to a single monitored entity (e.g. pressure and glycolic value sensors are associated to a patient in the health-care domain, humidity and temperature sensors are associated to a room in the environment monitoring case, etc.).

The alert notification system bridges monitored and monitoring components and is substantially responsible for:

1. identifying alarm events;
2. identifying the monitoring entities being able to manage the alarm event when it is generated;
3. forwarding data to the monitoring entity when requested.

As to item 1), we identified and codified three categories of alarm events: instantaneous, interval-based and multiparametric. Instantaneous alarms are triggered each time the value of a parameter leaves a pre-established range (e.g. humidity exceeds a certain threshold value). Interval-based alarms [8] are associated to situations lasting for a span of time (e.g. rapid huge increase of temperature). They are based on the knowledge of the dynamics of parameters and on the reasoning over their temporal evolution.

Multiparametric alarms are built by considering simultaneously several parameters, which, taken in isolation, are irrelevant, but which become meaningful if they are related with each other (e.g. during haemodialysis therapy, a slight increase in heart rate associated with a light increase in the systolic blood pressure may be symptomatic of a pathological state [9].

The diverse categories of alarms are characterized by values, thresholds, time windows, etc., which are necessary to detect the event. For instance, instantaneous alarms need threshold values, interval-based alarms need the definition of time windows and thresholds for the speed variation, multiparametric alarms need the list of parameters to be simultaneously observed, etc.

As to item 2), it is a complex task for several reasons. First of all, the available monitoring entities must be supposed to vary over time (consider for instance familiars and health operators in the health-care case, which may be available only in certain time intervals, which may be not necessarily rigorously predefined). Therefore, the system must support the dynamic registration and deregistration of the available entities, together with a rich advertising of their characteristics (e.g. the description of their capabilities: a familiar has less competence than a doctor to manage a severe alert).

Item 3) is the most simple to explain, as it substantially deals with the need of storing raw data and of allowing for bidirectional flux of data. The most onerous issues, in this case, are related to the amount of data to transmit, the availability of network bandwidth, of graphical utilities for visualizing data patterns, and so on. But such issues are outside the scope of this paper.

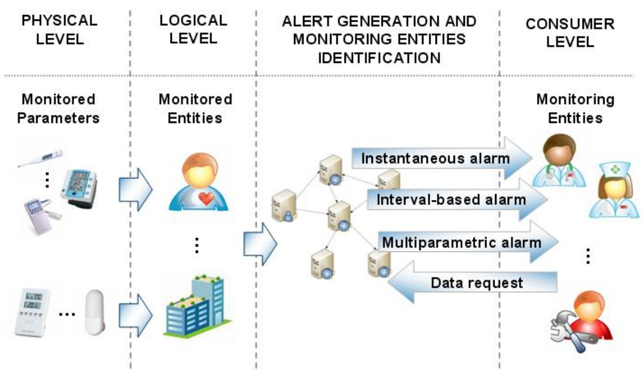

Figure 1. A schematic representation of alert generation systems

IV. System Logical View

A logical view of the system is provided in Figure 2 (see [4] for a more detailed description). The figure focalizes on the multi-agent system organization, but, as explained in the following, the other two components of the system, ontology and logic-based rules, have a fundamental role in shaping the system behavior. Agents have four fundamental roles:

Registration agents (RA) are in charge of managing agent registration and deregistration.

Context provider agents (CPA) get and filter raw context data and detect alarm events. Such agents are distinguished in physical CPAs (CPA$_p$), virtual CPAs (CPA$_v$) and logical CPAs (CPA$_l$) [10]. CPA$_p$ and CPA$_v$ agents manage data concerning a single context source. CPA$_p$ agents are responsible for identifying both instantaneous and interval-based alert events. CPA$_v$ agents are in charge of storing sensor data in suited archives (and periodically refreshing it), and of forwarding them to monitoring entities when requested. CPA$_l$ agents elaborate information provided by CPA$_p$ agents, so that context information coming from different context sources can be aggregated and multiparameter alarm events detected.

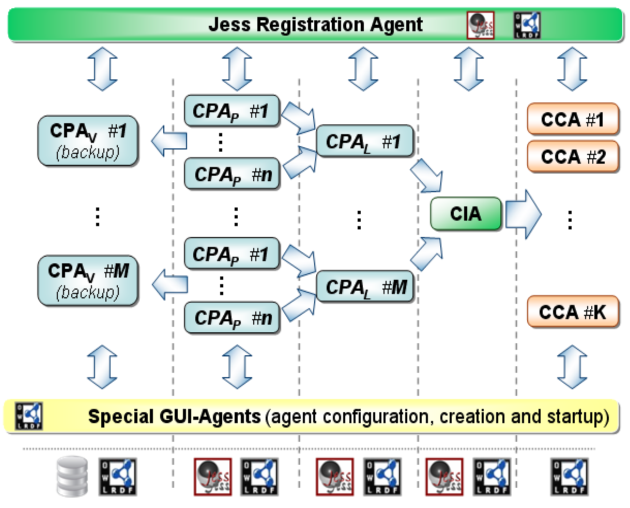

Figure 2. System Logical View

Context interpreter agents (CIA) are in charge of observing context changes sensed by CPAs, and, as consequence of these changes, of identifying the set of actions that should be performed. Substantially, they are responsible for identifying the monitoring entity best suited to manage an alarm situation, for contacting it and for forwarding context information to it.

Context consumer agents (CCA) are responsible for performing the actions triggered by CIAs. Actions provide the application reaction to context information changes, which may assume diverse forms, such as the generation of a signal, the delivery of a notification or a web service request. Moreover, they may request further data to CPAs by contacting the suited CPA$_v$ agent.

V. The Enabling Technologies

Agents were implemented by using the Java Agent Development Environment (JADE) [11], which supports the development of agent applications in compliance with the FIPA specifications.

The ontology, created by using the open source Protégé [12], was codified in OWL [13], as it is a key to the Semantic Web and was proposed by the Web Ontology Working Group of W3C.

The rule-based domain knowledge was implemented with Jess [14], chosen for its high computational speed and its good integrability with JADE.

Moreover, in order to integrate the three technologies and make an easy-to-customize system, we:

- adopted the Protégé BeanGenerator Plugin [15], which converts OWL ontologies into JavaBeans, and made an in-house adaptation of it in order to:
  - extend the range of supported types of data
  - automatically create Jess templates based on OWL classes
- imported and extended the so-called SimpleAbstractJadeOntology [16], which renders a knowledge base JADE compliant.

VI. A Reconfigurable System

System design was carried out by emphasizing commonalities between different application domains. As a result, the system is based 1) on a hierarchically structured ontology, delegating to lowest levels the conceptualization of elements specifics of the application domain, 2) on *normalized* rules [17], i.e. on rules amenable to be used in a variety of cases without any change to their structure, and 3) on general purpose agents, whose behavior is governed by ontology codification and rule-based reasoning. In the following, we show how these three implementation choices were carried out.

As to ontology, a common practise, when developing ontologies, is to adopt a *top level* (upper) shared conceptualization [18] on top of which domain ontologies are built. Top level ontologies codify general terms which are independent of a particular problem or domain. Once the top level ontology is available, several *lower level* ontologies can be introduced, with the scope of incrementally specializing concepts from the high level generic point of view of the upper ontology to the low level practical point of view of the application.

This way of implementing the ontology leads to a hierarchical architecture organized into three levels (Figure 3).

The top level concepts of our taxonomy contain terms useful for codifying the multi-agent environment thus facilitating their interoperation. Indeed, the top level ontology reflects the JADE codification of messages, as represented in the publicly available ontology named "OWLSimpleJADEAbstractOntology" [16].

The "context middle level ontology" codifies general concepts related to context, such as "MonitoredEntity" and "MonitoringEntity". Terms specifically related to the application domain are finally introduced at the bottom level (see Section 7).

High level classes of our top level ontology were used to codify normalized rules, which provide a general-purpose approach to different use contexts.

Normalized rules can substitute a set of rules having similar conditions and actions. In this way, the same code can be used in different use cases by simply changing the value constraints of the normalized rule. For example, rules such as:

"if Body Temperature for the Monitored Patient exceeds Maximum Body Temperature and if contemporarily the Heart Rate for the same Monitored Patient exceeds Maximum Heart Rate threshold, then generate a Medium Gravity Alert Event and contact an available Duty Doctor"

and

"if Relative Humidity for the Monitored Room goes below the Minimum Relative Humidity threshold and if at the same time Room Temperature exceeds a specified maximum threshold, then generate a Low Gravity Alert and contact an available technician"

can be substituted by the following normalized rule:

"if, at time T, parameters $P_{X,\ldots,Z}$ for monitored entity E_X exceed their thresholds $Th_{X,\ldots,Z}$ AND these events are classified with a certain level G of gravity AND there is a monitoring

entity U_X available at the same time T, THEN generate an alert A with a certain level of gravity G AND forward the alert to a proper monitoring entity U_X"

As shown above, when normalized rules are not adopted, each kind of parameter, threshold, etc. corresponds to a different rule. As a result, the amount of rules to be codified becomes very huge, with a large effort in codification and maintenance activities.

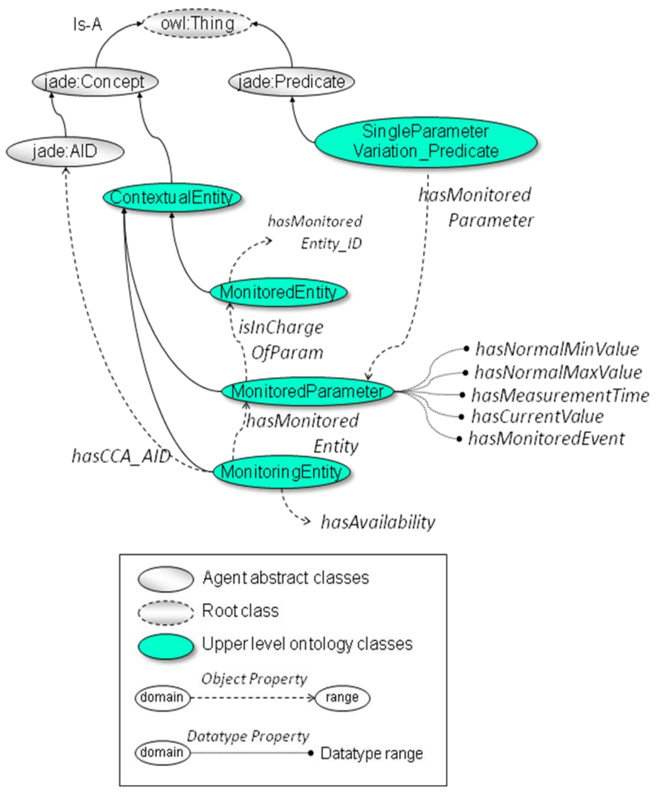

Figure 3. Upper Level ontologies

As to agent design, agents have a common structure, which has to be customized by defining a suited *reasoning template* and by providing some configuration information at start-up. Indeed, agent role is always that of transforming some sort of low-level context information into a form of higher level context information, based on the reasoning template. This kind of conversion is made in an incremental fashion (Figure 4), with the final global result of forwarding meaningful high level information to appropriate end-users.

For instance, CPA_p convert raw data (low-level information) received by physical sources into alarms (high level information) provided that they are informed about which thresholds generate which kind of alarm (reasoning template). Similarly, CPA_l convert knowledge coming from several CPA_p (low-level information), into aggregated alarms (high level information) provided that they are informed about patterns and thresholds (reasoning template). Finally, CIA convert alarm information coming from CPA agents into meaningful messages for end-users provided that they are informed about the appropriate correspondence between the different typologies of alarms and the different kinds of end-users.

As to agent inner structure, it consists in the continuous reasoning over context information and on the consequent generation of messages (Figure 5). Context information instantiates Java classes and asserts facts in the Jess working memory (WM). As a result, rules are fired. This induces a further instantiation of agent classes, upon which the agent builds the message to be forwarded to other agents.

Both agent classes and rules operate on ontology entities. As explained more in detail in the following section, this allows them to reason over different kinds of real-life contexts, provided that the upper-level ontology is suitably extended and provided that configuration data, codified in a simple format, is given. This is possible thanks to our strong effort toward integration between ontology, rule-based and agent technologies.

In order to simplify our explanation and to provide concrete validation results, since now on we base our narration on two different well-known real-life domains: health-care and environment monitoring.

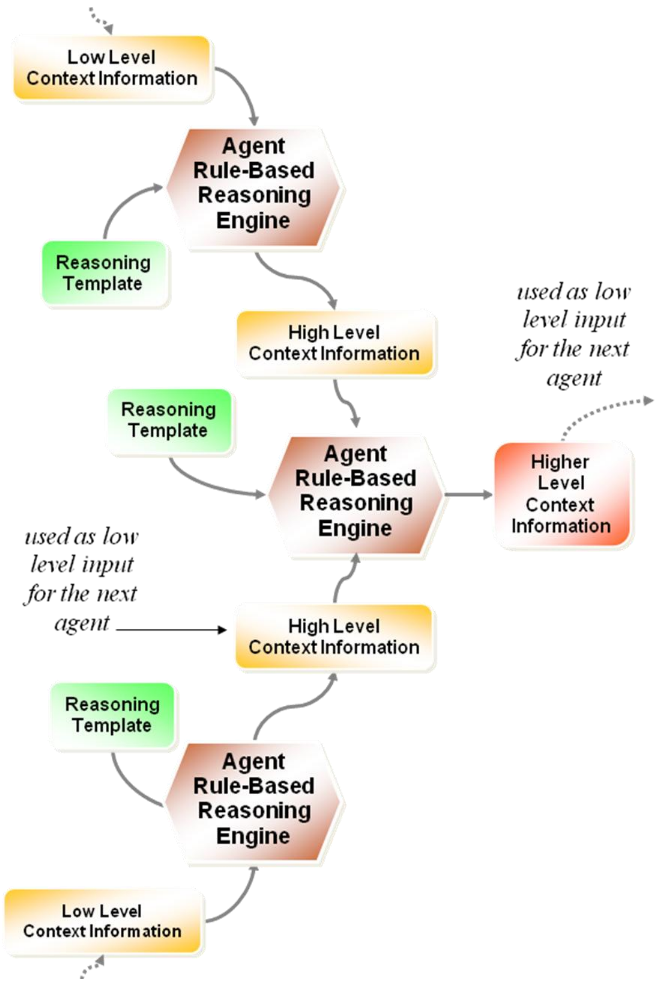
Low Level Context Information
Agent Rule-Based Reasoning Engine
Reasoning Template
High Level Context Information
used as low level input for the next agent
Reasoning Template
Agent Rule-Based Reasoning Engine
Higher Level Context Information
used as low level input for the next agent
High Level Context Information
Reasoning Template
Agent Rule-Based Reasoning Engine
Low Level Context Information

Figure 4. Agent basic structure and incremental context data transformation

Suppose therefore to consider the following health-care use case:

One or more patients are continuously monitored by medical equipment. An alarm may be generated when patient data exceed some threshold, when different parameters (such as temperature or glycolic value) are simultaneously over some other threshold, and/or when the speed with which the variation occurs is high. Information about patients *(thresholds, history, etc.) and doctors (specialization, availability, telephone numbers, etc.) is used to identify the doctor best suited to manage an alarm situation. The doctor is alerted on his laptop/mobile and may request further information to best analyze the patient situation.*

and the indoor use case reported below:

One or more indoor regions (hospital rooms, house sections, laboratories…) are continuously monitored by sensing equipment in order to maintain a set of life conditions or storage requirements. An alarm may be generated when sensed data exceed some threshold, when different parameters (such as room temperature or relative humidity) are simultaneously over some other threshold, and/or when the speed with which the variation occurs is high. Information about those physical regions (thresholds) and technical operators (availability, telephone numbers, etc.) is used to identify the technician best suited to manage an alarm situation. The operator is alerted on his laptop/mobile and may request further information to best analyze the room situation.

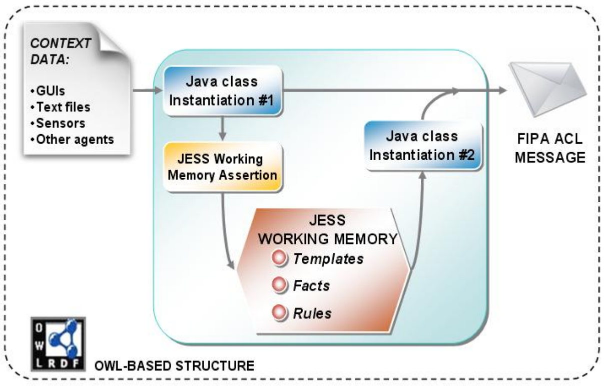

Figure 5. Agent inner structure

VII. The Customization Effort: before Compilation

The system customization phase is in many cases very easy, being limited to the extension of predefined ontology classes. More in detail, system customization requires the execution of a sequence of activities, most of which have been automated. Some customization activities must necessarily be executed before system compilation, as they completely "shape-up" the system, others must be executed during the start-up phase.

Before system compilation, we have to:

1. *customize the ontology*, i.e. extend the ontology so that classes specific to the environment are added;

2. *customize the agents*, i.e. create the Java classes corresponding to the added ontology entities;

3. *customize the rule-engine*, i.e. create the Jess templates corresponding to the added ontology entities.

As to Activity 1, before launching the system we have to verify whether the entities of the monitored system (i.e. parameters, monitored entities and monitoring entities) are ontologically codified or not. In the former case, they are listed in the start-up user interface, otherwise the entities have to be codified into the ontology.

Figures 6 and 7 show how the upper level ontology was extended respectively to the health-care and environment monitoring cases. As shown in the figures, ontology extension often corresponds to the insertion of new class names, as the datatype and object properties needed by the system have already been codified in the superclass.

Once that the ontology has been extended, activities 2 and 3 are automatically performed by an in-house extension of Bean Generator which updates rules and Java classes in order to manage the added concepts. In other words, users are requested only to set-up input data and to extend the ontology, if needed. Then, the system configuration phase can start.

VIII. The Customization Effort: at start-up

At start-up, it is necessary to define agent reasoning templates, which govern agent behavior. This phase was made easy by implementing an ad-hoc user-friendly graphical user interface. Reasoning templates depend on the kind of agent they are applied to. They define agent relationship with other entities (e.g. CPA_p agents are informed about the monitored parameter and the monitoring entity they are in charge of) and provide basic information agents need to operate.

For instance, CPA_p agents need the thresholds for generating alarms, whilst CPA_l also need the list of parameters they have to aggregate.

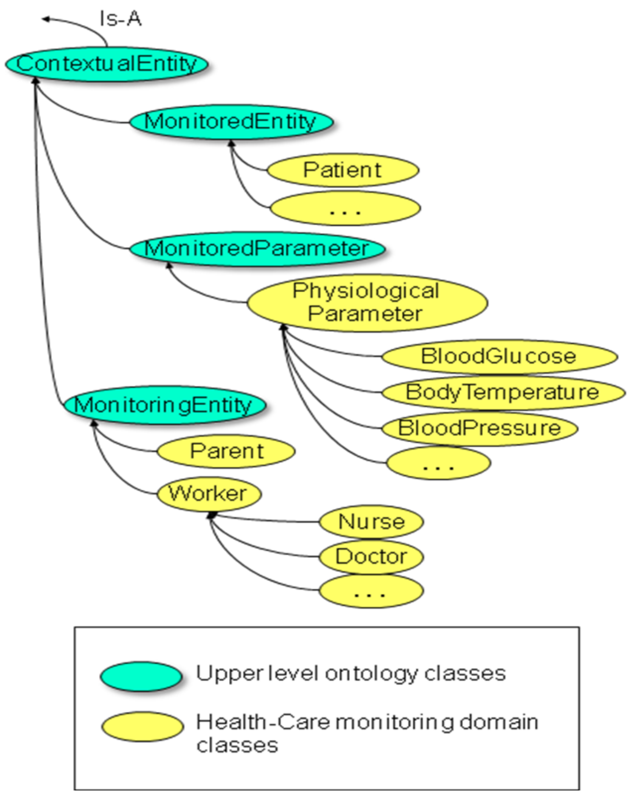

Figure 6. Ontology extension to the health-care domain

Finally, CIA agents need to be informed about the kind of end-user to be contacted in case of alarm (for instance a familiar is enough in case of low-level alarm related to blood pressure, whilst a cardiologist is needed in case of high level aggregated alarm).

Jess templates and Jess normalized rules are automatically generated on the basis of such configuration information. The following code snippet shows a CPAl rule (valid in both the health-care case and in the environment monitoring case), useful for generating multiparameter alerts, which was automatically generated on top of a reasoner template:

```
(defrule aggregate-temporal-variations-for-2-params
 ?f0 <- (SingleParameterVariation_Predicate
        (hasParamName ?pn0)
        (hasParamMeasurementTime ?t0))
 ?f1 <- (SingleParameterVariation_Predicate
        (hasParamName ?pn1)
        (hasParamMeasurementTime ?t1))
 ?f-check <- (aggregation-pattern-for-2-params
       (pattern-time-width ?ptw &: (<= (abs(-?t0 ?t1))
?ptw)) (param-0 ?pn0) (param-1 ?pn1))
=>
(send-aggregate-alert)
(retract ?f0) (retract ?f1))
```

This rule is fired when two facts, corresponding to instantaneous alarms, are stored in memory in the same time window. The function *send-aggregate-alert* sends the alert event to the CIA agent.

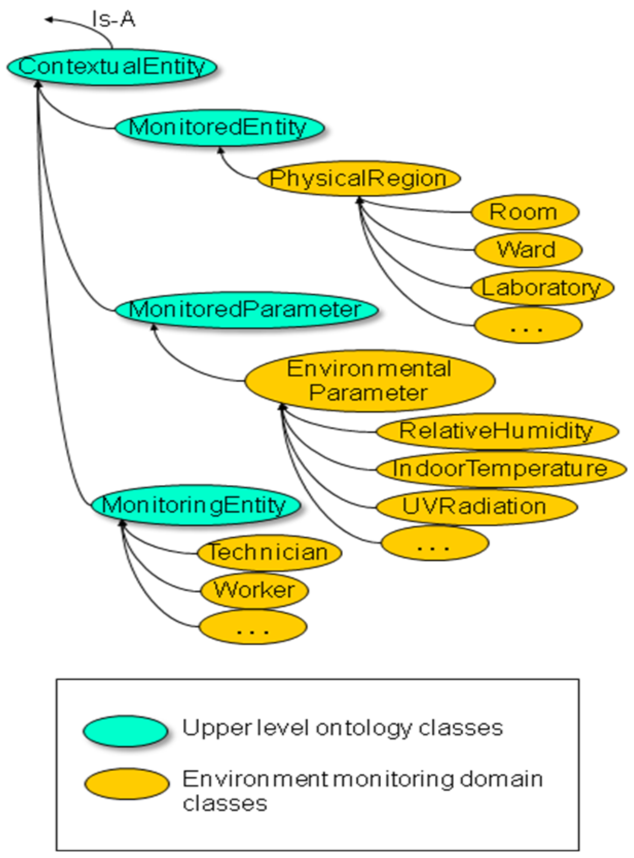

Figure 7. Ontology extension to the environment monitoring domain

IX. System Operation

In order to briefly describe system operation, let us focus on CPA_P agents, which identify both instantaneous and interval-based alert events. They are continuously fed with sensor data, which instantiate a predicate class (see Figure 5, *Java class instantiation #1*), namely the SingleParameterVariation_Predicate class, whose instantiation for the health-care use case (BodyTemperature parameter) is:

```
SingleParameterVariation_Predicate
  hasParamName: BodyTemperature
  hasParamCurrentVal: 38.5
  hasParamNormalMaxVal: 37.0
  hasParamNormalMinVal: 35.5
  hasParamMeasurementUnixTimestamp: 1255461262
  hasMonitoredEntitySourceID: 3
  hasMonitoredEntityType: Patient
```

the instantiation for the environment monitoring use case (RelativeHumidity parameter) is instead:

```
SingleParameterVariation_Predicate
  hasParamName: RelativeHumidity
  hasParamCurrentVal: 85.0
  hasParamNormalMaxVal: 60.0
  hasParamNormalMinVal: 30.0
  hasParamMeasurementUnixTimestamp: 1255462864
  hasMonitoredEntitySourceID: 5
  hasMonitoredEntityType: Room
```

Once that the Java bean has been instantiated, it is asserted into the Jess WM (see Figure 5, *Jess Working Memory Assertion*). Instantaneous variations are tackled by using the following normalized rule, which compares the sensed value with thresholds:

```
(defrule verify-sensor-data
   ?f <- (SingleParameterVariation_Predicate
       (hasParamName ?pn)
       (hasParamCurrentVal ?par-c |: (> ?par-c ?par-max) |:
(< ?par-c ?par-min))
       (hasParamNormalMaxVal ?par-max)
       (hasParamNormalMinVal ?par-min)
       (hasParamMeasurementTime ?par-time))
     =>
    (send-sensor-alert))
```

The Java user function *send-sensor-alert* notifies the alarm event to CPA_L. Interval-based events are detected by using a similar procedure. A normalized rule detects the occurrence of facts concerning the same parameter and having been asserted in the same time window. If the rule fires, a Java class containing all the meaningful information concerning the detected event is instantiated (see Figure 5, Java Class instantiation #2) and its content forwarded to the CPA_L.

X. Experimentation

The overall system was tested within a network of computers connected by a TCP/IP-based 100Mb/s network. Computers run under Linux operating system and have the following characteristics:

1. CPU: Intel Pentium, 3.00 GHz (single core); RAM: 512 MB;
2. CPU: AMD Athlon, 1.8 GHz (single core); RAM: 478 MB;

The experimentation consisted in collecting average alarm generation and transmission times in diverse configurations (Figure 8) of the health-care use case.

The simplest consists of a single entity monitored by one sensor and assisted remotely by four monitoring entities.

The most complex consists of 5 entities, each being monitored by 4 sensors and being assisted by 4 monitoring entities. Results concern average times obtained over 30 runs.

The system monitors data sensed by sensors, and, based on monitored entity info (loaded once at system start-up) determines alarm events. Alarm events trigger actions which basically consist in identifying the best suited monitoring entity (i.e. the doctor having the appropriate specialization and being available in the nearest time interval) and contacting him/her (via mobile).

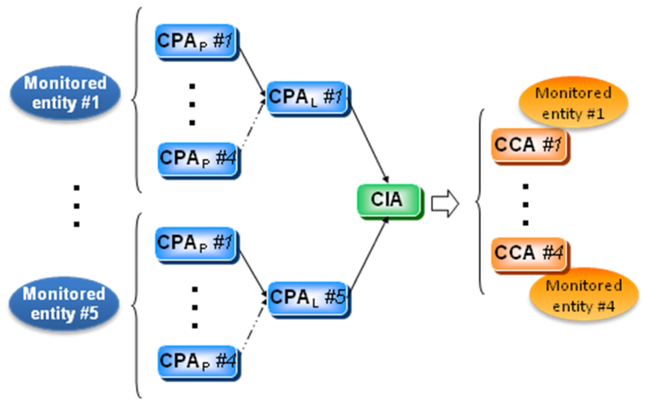

Figure 8. Experimentation set-up and configurations

Measured times were taken by imposing two “stressing conditions”: 1) each sensed parameter has always out-of-range values, 2) all sensors send data at the same time. They consist in:

- the time elapsing at the CPA$_p$ side between the reception of a datum from the sensor and the triggering of the corresponding alarm obtained by means of rule-based reasoning;
- the time elapsing at the CPA$_l$ side between the reception of a message coming from CPA$_p$ and the triggering of the corresponding alarm obtained by means of rule-based reasoning;
- the time elapsing at the CIA side between the reception of a message coming from CPA$_l$ and the forwarding of a message to CCA.

As shown in Table I, CPA$_p$ are responsible for filtering data coming from a single sensor, therefore their times do not change when the number of sensors and/or of monitored entities change. Each CPA$_l$, instead, elaborates data coming from sensors connected to a single monitored entity. In our experimentation set-up, they elaborate data coming from up to four sensors and produce two kinds of aggregate alarms: the former associates “BodyTemperature” and “BloodGlucose” patterns, the latter combines “BloodPressure” and “HeartRate” patterns. Therefore, CPA$_l$ times increase with the number of sensors attached to single patients. CIA times, finally, depend both on the number of sensors and on the number of patients (i.e. both on the number of CPA$_p$ and on the number of CPA$_l$).

Table I also illustrates data obtained by aggregating the average times obtained for each configuration. As shown in the table, processing times are very good, thus demonstrating the amenability of the proposed approach to attach also more complex monitoring problems.

Agent	(# Sensors, # Patients)					
	(1,1)	(2,1)	(2,2)	(4,1)	(4,2)	(4,5)
CPA$_P$	3.68	3.68	3.68	3.68	3.68	3.68
CPA$_L$	/	4.74	4.74	5.96	5.96	5.96
CIA	2.24	2.87	2.93	3.28	6.97	24.49
Ttot	5.92	11.29	11.35	12.92	16.61	34.13

Table 1. Performance results [ms].

XI. Conclusion

In this paper, a smart re-configurable system suitable for monitoring purposes and able to take decisions coherently with the context data has been presented. The system harmonizes heterogeneous technologies, such as agents, ontologies and rule-based inference engines. As a result, the ontology provides the knowledge codification needed to support both agent reasoning and communication. The effectiveness of the system is demonstrated by using simple and concrete examples in two real life scenarios. The very promising results, in terms of easiness of use and reconfigurability of the system, make the proposed approach a very good candidate for the solution of complex monitoring problems.

References

[1] R. Goebel, et al., “ Lecture Notes in Artificial Intelligence”. *Proceedings of 12th Conference on Artificial Intelligence in Medicine*, AIME 2009, Verona (Italy), July 18-22, Springer.

[2] U. Cortés and M. Poch, Editors, *Advanced Agent-Based Environmental Management Systems*. Whitestein Series in Software Agents Technologies and Autonomic Computing. Birkhäuser Verlag, Berlin, 2009.

[3] F. Paganelli, E. Spinicci and D. Giuli, “ERMHAN: a context-aware service platform to support continuous care networks for home-based assistance” *International Journal of Telemedicine and Applications*, Vol. 2008, Issue 5.

[4] A. Esposito, L. Tarricone, M. Zappatore, L. Catarinucci, R. Colella and A. Di Bari, "A Framework for Context-Aware Home-Health Monitoring". *In UIC '08 Proceedings of the 5th international conference on Ubiquitous Intelligence and Computing*, pp.119-130, Oslo, Norway.

[5] X. Wang, J.S. Dong, C.Y. Chin, S. Hettiarachchi and D. Zhang, "Semantic Space: An Infrastructure for Smart Spaces", *IEEE Pervasive Computing***,** Vol.3, No.3, pp. 32 – 39, 2004.

[6] T. Gu, H.K. Pung and D. Zhang, "A service oriented middleware for building context-aware services", *Journal of Network and Computer Appications (JNCA)*, Vol. 28, Issue 1, pp. 1-18, Elsevier, 2005.

[7] H. Chen, T. Finin and A. Joshi, "An ontology for context-aware pervasive computing environments", *The Knowledge Engineering Review*, Cambridge University Press, Vol.18, pp.197–207, 2003.

[8] J.C. Augusto, "Temporal Reasoning for Decision Support in Medicine". *Artificial Intelligence in Medicine*, v.33, n.1, pp.1-24, Elsevier B. V., 2005.

[9] A. Otero, P. Felix, F. Palacios, C. Perez-Gandia and C.O.S. Sorzano, "Intelligent alarms for patient supervision", *Proceedings of the IEEE International Symposium on Intelligent Signal Processing*, WISP 2007, pp.1-6, 2007.

[10] J. Indulska and P. Sutton, "Location management in pervasive systems", *CRPITS'03 Proceedings of the Australasian Information Security Workshop*, pp.143–151, 2003.

[11] Jade, 2010 http://jade.cselt.it (last access: February 2011)

[12] Protégé, 2010 http://protege.stanford.edu/ (last access: February 2011)

[13] OWL, 2010 http://w3.org/TR/2004/RDC-owl-features-20040210/ (last access: February 2011)

[14] E. Friedman-Hill, E. *Jess In Action*. Manning.Publications Co., Greenwich, CT, USA, 2003.

[15] BeanGenerator, 2010 http://protege.cim3.net/cgi-bin/wiki.pl?OntologyBeanGenerator (last access: February 2011)

[16] CLOnto, 2010 http://jade.tilab.com/doc/tutorials/ CLOntoSupport.pdf (last access: February 2011)

[17] J. Williamson, 2010. http://www.jessrules.com/ jesswiki/view?KeepYourRulesNormalized (last access: February 2011)

[18] N. Guarino, "Formal Ontology and Information Systems" In N. Guarino, editor, *Proceedings of the 1st International Conference on Formal Ontologies in Information Systems*, FOIS'98, Trento, Italy, pages 3--15. IOS Press, (1998).

Author Biographies

Luciano Tarricone is a Full Professor of Electromagnetic Fields at the University of Lecce, Italy. He received his laurea degree (with honors) from the University of Rome "La Sapienza", Italy, and his PhD from the same university, both in Electronic Engineering. In 1990 he was a researcher at the Italian National Institute of Health, and between 1990 and 1994 at the IBM European Center for Scientific and Engineering Computing. Between 1994 and 2001 he was at the University of Perugia, Italy. Since 2001 he has joined the University of Lecce. His main research areas are: supercomputing and knowledge engineering for Electromagnetics, environmental electromagnetic compatibility, CAD of microwave circuits and antennas. He authored around 300 papers in international conferences and journals, and edited 3 volumes in the area of high performance computing and knowledge engineering for electromagnetics.

Alessandra Esposito is a free-lance consultant in the area of Computer Science and Information Technologies, with a focus on networking, web and grid applications for research in universities and small, medium and large companies. She received her laurea degree (with honors) in Electronic Engineering from the University of Naples. Between 1990 and 1994 she was with IBM Scientific Center in Rome, Italy. In 1994 and 1995 she was a system engineer for Sodalia, Trento, Italy, involved in research and development in the area of distributed systems. Since 1995 she has cooperated with several research institutions, universities and business companies, in the framework of educational, research and industrial projects. She authored about 90 papers in international conferences and journals. She is co-author of the book "Grid Computing for Electromagnetics" edited by da Artech House.

Marco Zappatore is a PhD Student in Information Engineering at the University of Lecce. He graduated in Information Engineering at the same University in 2005 with a thesis dealing with an hybrid genetic algorithm for 3G wireless network optimum planning. In 2008, He received a Laurea Specialistica degree (2nd level) in Telecommunications Engineering from the same University with a thesis concerning the EM enabling technologies for software intelligent systems in the healthcare domain. He currently collaborates with the Electromagnetic Fields Group at University of Salento. His research activities are mainly focused on: Semantic Web, Context-Awareness and Multi-Agent Systems applied to the EM area, optimization techniques for wireless networks planning. He co-authored about 20 papers in international conferences and journals.