Smart Environment Software Reference Architecture

Conference Paper · January 2009

DOI: 10.1109/NCM.2009.115 · Source: doi.ieeecomputersociety.org

CITATIONS
16

READS
595

4 authors:

Alejandro Fernández-Montes
Universidad de Sevilla
50 PUBLICATIONS **305** CITATIONS
SEE PROFILE

Juan A. Ortega
Universidad de Sevilla
190 PUBLICATIONS **911** CITATIONS
SEE PROFILE

Juan Antonio Alvarez-Garcia
Universidad de Sevilla
75 PUBLICATIONS **630** CITATIONS
SEE PROFILE

Luis Gonzalez-Abril
Universidad de Sevilla
247 PUBLICATIONS **1,195** CITATIONS
SEE PROFILE

2009 Fifth International Joint Conference on INC, IMS and IDC

Smart Environment Software Reference Architecture

A. Fernandez-Montes, J. A. Ortega, J. A. Alvarez
Department of Computer Science
University of Seville
Seville, Spain
{afdez,jortega,jaalvarez}@us.es

L. Gonzalez-Abril
Department of Applied Economics
University of Seville
Seville, Spain
luisgon@us.es

***Abstract*—Nowadays ubiquitous computing is spreading to all scopes of our lives. Smart environments are present at every location such us homes with automation and control devices, offices full of control networks to assist workers, or hotels with even more control devices in order to save energy and satisfy guests preferences.**

This paper focus on the proposal of a Reference Architecture for developing Smart Applications and deploy them in Smart Environments. The proposal consider three main process in the Software Architecture of these applications: *a*) perception, *b*) reasoning and *c*) acting.

***Keywords*-Smart environments, software architecture, ubicomp.**

I. Introduction

We can define a smart environment as *one that is able to acquire and apply knowledge about the environment and its inhabitants in order to improve their experience in that environment* [1].

Smart home technologies are often included as a part of ubiquitous computing. Mark Weiser [2] outlined some principles to describe Ubiquitous Computing (ubicomp) from which we emphasize that the purpose of a computer is to help you do something else.

Home technologies have tried to help home inhabitants since its creation. Nowadays, due to the popularization of computational devices, ubiquitous computing is called to be the revolution to develop smart systems with artificial intelligence techniques.

Domoweb [3] was a research project originally developed as a residential gateway implementation over the OSGi (Open Services Gateway Initiative) service platform. The experiences obtained from this project allow us to face an even more challenging project, the proposal of a General Software Reference Architecture to develop smart applications where all the components of a smart environment can interact flawlessly and reach automatism objectives.

Typical components of a smart environment are widely studied over the literature, but we can emphasize the approach of D.J. Cook, and S.K. Das at [4] which shows a general organization of these components. These are divided in four layers: *a*) physical, *b*) communication, *c*) information and *d*) decision. This approach joins hardware with software agents, so very heterogeneous elements appear in the same component model such as a *decision maker* and *sensors* or *actuators*. All these components must collaborate to achieve the goals of automatism that a smart environment is required. This is the main motivation of current work, so a Reference Software Architecture is proposed.

Other interesting approaches have been proposed. Costa [5] presents a model where each issue is matched with the characteristics that address it. Weis [6] shows a high level programming language for rapid prototyping of pervasive applications. The approaches from Rehman [7] and Bannach [8] are focused in interactive context-aware applications and prototyping of activity recognition applications respectively. In contrast with these approaches, ours is a General **Software** Architecture for Smart (pervasive) Applications.

II. Reference Architecture

The proposal of a General Reference Software Architecture to develop smart applications is based in the goals of the ubiquitous computing subject proposed by Weiser [2]. Taking this as a starting point, we believe that automation in smart environments should be organized as a continuous interaction between three main tasks: *a*) perception, *b*) reasoning and *c*) acting. (see figure 1).

The perception of the state of the environment is performed by means of the physical components distributed through out the environment, reasoning must be done about the state and it produces possible actions to take, finally these actions must be carried out.

III. Perception

The perception process should be divided in different tasks in order to provide an accurate *perception* of the real world (see figure 2).

It has to deal with low-level details to retrieve data from real world and adapt it to a knowledge base which must agree the smart environment model proposed in [9]. This process have to clear this data of erroneous, insignificant or redundant values in order to achieve the accuracy required by next process.

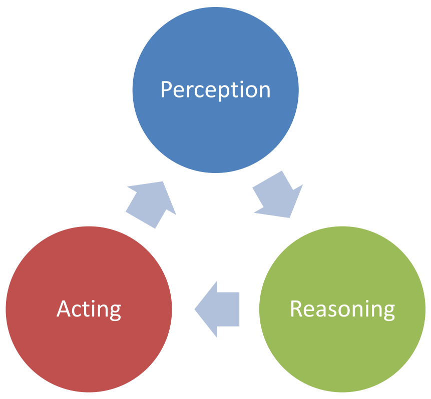


Figure 1. The cycle of the automation process in a smart environment.

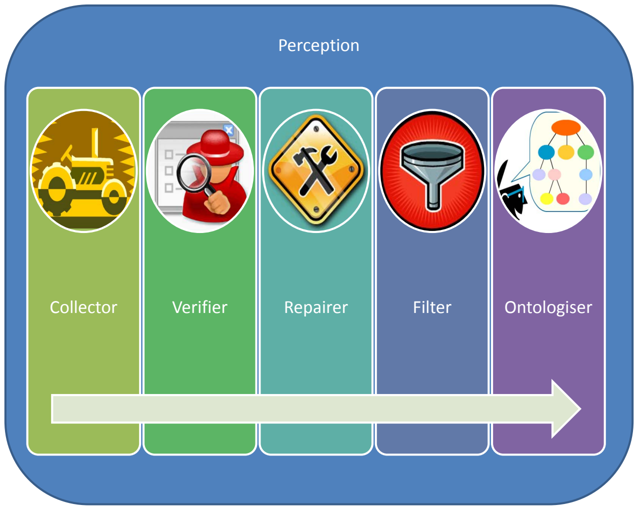


Figure 2. Tasks of the perception process.

A. *Collector*

This is the lowest-level task, so its main objective is to retrieve data from physical devices. It will usually have to deal with gateways devices of every technology deployed over the smart environment.

The properties of the environment suitable to be captured by these sensors are listed in [9]. Actually a subset of these properties could be selected to achieve any particular application or automation process (e.g. put lights off when nobody is at home does not need conditioning information from the environment).

Sensors must be deployed in an organized manner to avoid redundant or insignificant information and reduce costs. When developing the *Data Collector* we have to consider if devices should be programmed or not (or if both types of sensors are present in the environment).

Majority devices are pre-programmed and can't extend its basic functionality (e.g. X-10 Motion sensor), so developers have to adapt to them. On the other hand more and more devices have the ability to extend or modify their performance (e.g. Sentilla *Tmote*).

The latter are much more challenging for developers due to they can be adapted to current needs for a concrete application or circumstances. In this case, Data Collector task covers daemons developed to retrieve information and small applications running on devices as well, so both elements have to agree what information is sent, periodicity of these requests and so on.

B. *Verifier*

The main purpose of the *Verifier* task (as its name means) is to *verify* that the data (which is currently being received by the *Data Collector*) is correct. But the challenge now is how the *Verifier* can determine if the data is correct or not. There is no unique solution for all possible environments, so the verification has to adapt to each environment dynamically.

We propose the *Verifier* to maintain a rule engine where verification rules can be deployed, modified and used in order to determine where data is right or wrong for current environment. The rule engine will have to offer programmers a flexible way to read, add, modify and delete rules.

By the way this task has to work side by side with the *Repairer* when incorrect data is received in order to fix invalid data. The mechanism of communication between the *Verifier* and the *Repairer* must be determine. Typical implementations of this mechanism would be the publication of a *repair service* by the *Repairer* that will be invoked by the *Verifier*.

We can conclude the *Verifier* can be seen as a filter applied over all the data received, and it can be used to reject data for any reason (incorrect, redundant and other reasons).

C. *Repairer*

The *Repairer* task will try to fix incorrect data detected by the *Verifier*. The corrections applied to data can be done in very different ways such us:

- **Ignore data**. First option is to ignore data, it means to set it with unknown value. The *Ontologiser* will save that value as required by storage system (e.g. Weka-arff '?' character, or SQL null value).
- **Adjust data**. If a value is incorrect, but its distance with a correct value is less than a threshold (previously customized) the value could be adjusted to nearest correct value.
- **Replace data**. Other option is to replace data with previous correct data. (e.g. if temperature sensor returns 100 Celsius, and previous data is 25, we could replace current value with 25).
- **Reject data**. If current values are not suitable to be repaired, the repairer will reject it.

Once a set of data has been received, verified and repaired, it has to be sent to the *Ontologiser* to be organized and stored.

D. *Ontologiser*

Artificial intelligence algorithms need a solid base of knowledge to work. The main goal of the Ontologiser is to organize data to conform a model of the real world.

This fact demands us a great effort for building a model of a smart environment. Other studies have helped us to compose the model [10] [11] [12] which has been arranged in four main categories explained in [9].

Synchronization. *Ontologiser* will have to synchronize data from a world full of asynchronous devices and events. Automation applications usually need sets of data composed from multiple devices values. Data from different devices must by synchronized for a latter aggregation, so this task will have to define the rules to synchronize values from multiple and very heterogenous sources.

Implementations of the synchronization process will vary depending on the goals of a concrete smart application, but all implementations will share some elements such as:

- **Data buffer** will store data received that is waiting to be paired with other data.
- **Garbage collector** will supervise the size of the buffer, and will periodically clean the buffer from data that couldn't be paired.

Aggregation. Once data is synchronized it must be joined in a set of data for a concrete application. Each attribute should conform the Smart Environment Model [9].

When data is aggregated it is ready to be stored in the knowledge base that feeds the reasoning tasks and learner process. The knowledge base format will be any of the *de facto* standards like arff, or any relational database.

IV. Reasoning

Reasoning process in smart environments can be separated in several tasks (see figure 3) which interact to achieve three main goals: *a*) learn, *b*) reason and *c*) predict.

Learning can be done through any of the techniques commonly known and widely studied in the literature. Attending to the scope of this work we can separate these techniques in two main categories: *a*) rule based and *b*) *black-box* based .

Rule based algorithms offers the advantage to be readable and easily understandable by humans. Rule-based techniques are studied in [13] and applied to automation of smart environments in iDorm [14] or combined with temporal reasoning in ADB [15].

On the other hand *black-box* based techniques don't offer a comprehensible model of the knowledge that is being learned, but these techniques have been successfully used in pattern recognition and learning in general. Neural Networks or Support Vector Machine are studied as a general technique by authors like [16], [17], [18] and applied to automation of smart environments in the Neural Network House [19], House_n [20] and Mav Home [10].

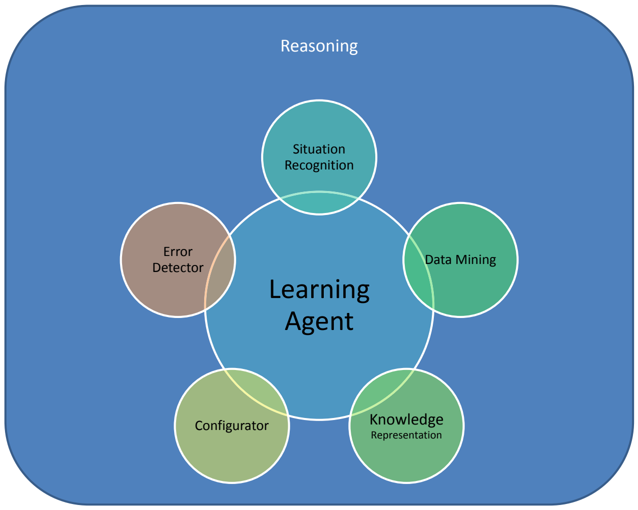


Figure 3. Typical tasks of the reasoning process.

A. *Learning Agent*

This task can be seen as the glue that joins any other task proposed to achieve the objectives of the Reasoning process. It has to deal with:

- *Data Mining* task to extract patterns,
- *Situation Recognition* task to label patterns recognized,
- *Prediction* task to find future actions to be taken and
- *Error detector* task to detect wrong decisions and to make related improvements over the entire process.

The learning can be carried out due to the dependence of the user preferences with his habits. For instance normally we have the same lighting preferences in our offices so these preferences must be learned at every environment, due to its dependence with the location, orientation of the window, devices setup and so on.

Learning techniques. Two families of algorithms, related with learning machine, can be considered although both are going to be supervised.

Support vector machine (SVM) and Neural networks (NN) are some of the techniques we can use to learn user lighting preferences. The main advantage of these techniques is that always offer an output. On the other hand it is hard to understand their outputs which is an important feature that prediction algorithms should implement as expounded in [9].

The other family of algorithms considered are the machine learning techniques based on rules. These algorithms provide

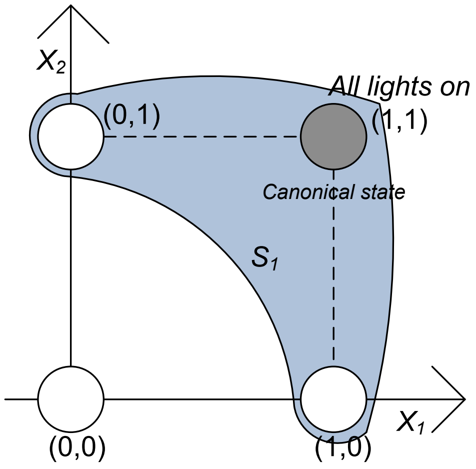


Figure 4. States for variables x_1 and x_2

an output easier to understand and interpret, but their main disadvantage is that if no rule matches current state, these algorithms don't offer an output.

B. Data Mining

Data mining task works side by side with the learning agent to extract information and find patterns from the data stored by the *Perception* (see section III) process.

Smart Environment Vectorization Data mining is fed with a stream of data with a concrete format. This is usually so-called the vectorization of data. This fact forces every concrete smart application to propose a vectorization of the environment and configure all the reasoning process with it.

C. Situation recognition

A situation is defined as a set of states which have similar values. This set can be defined by:

- a canonical state as a representative state of all the states in the set
- a distance function which define how to measure the difference between two states
- a threshold which define the maximum distance between the canonical state and the states of the set.
- a weight for each element of a state between [0,1]. A weight of zero represent that an element must not be considered.

The following example illustrates the definition of a situation. Let x_1 be the variable which represents the state of a light, and let x_2 be the variable which represent the state of another light. These variables are discrete, and its range is $0, 1$ which represent the states off and on respectively.

The figure 4 shows all the states that these variables can constitute.

We would like to define the situation S_1 where any of the lights is on. The canonical state of S_1 is $\{1, 1\}$ (all lights on)

Table I
STATES THAT BELONG A SITUATION.

State		Canon. State		Weights		d	Thresh.	$\subset S_1$
x_1	x_2	y_1	y_2	w_1	w_2			
0	0	1	1	1	1	2	1	no
0	1	1	1	1	1	1	1	yes
1	0	1	1	1	1	1	1	yes
1	1	1	1	1	1	0	1	yes

for the variables x_1 and x_2 respectively. A typical distance function could be defined as:

$$d([x_1, x_2], [y_1, y_2], [w_1, w_2]) =$$

$$= \sqrt{(x_1 - y_1)^2 * w_1 + (x_2 - y_2)^2 * w_2}$$

Where $[x_1, x_2]$ represent a state, $[y_1, y_2]$ represent the canonical state and $[w_1, w_2]$ represent the weights of x_1 and x_2 respectively which should be $[1, 1]$ for the situation S_1 any of the lights is on due to both lights should be considered. The threshold can be set to 1.

Table I shows the set of states that belong to the situation. If the distance is less or equal to the threshold, the state belongs to the situation. In other cases we can conclude the state will not belong to the situation.

D. Prediction

The knowledge acquired during the process of learning of the agent will be useful to make predictions about future actions that should be taken in order to achieve any automatism objective. These actions can be considered as the output of smart algorithms as explained in [21].

Desirable features of prediction algorithms. The features that a smart home system must implement are listed, specially related with the prediction algorithms the systems may have. So we don't focus on artificial intelligence techniques like the studies of [11], [22], [23] do, but on what are the most important and indispensable features that must be considered to develop prediction algorithms for smart environments which are briefly studied in [9] and just listed below: *a*) Prediction supported by last events and states, *b*) Predictable by the inhabitants/users, *c*) Understandable decisions, *d*) Wrong decisions detection and related improvement, *e*) Anomalies detection, *f)* Quick response when required and *g*) General policies management.

E. Error detector

The main purpose of this task is to supervise the actions taken by the inhabitants in order to detect opposite decisions between the machine and them. Therefore it will have to keep last actions taken by the smart environment and compare them with actions taken by the inhabitants.

The comparison of these actions is usually a complex operation where lots of elements take part into, and it has to take into account how many actions are going to be considered to make the comparison, how long these actions are going to be considered, or if actions over different devices could be opposite (e.g. open the window could be opposite to switch off the lights). All these aspects make the detection of wrong decisions much more challenging.

V. Acting

Finally, to close the cycle, Smart Environments have to act automatically to achieve a concrete smart application. This is the main purpose of the process of *Acting*. As shown in figure V the decisions and concrete tasks ordered by the *Reasoning* process have to pass through three main tasks *a*) policy manager, *b*) task scheduler and *c*) task runner1.

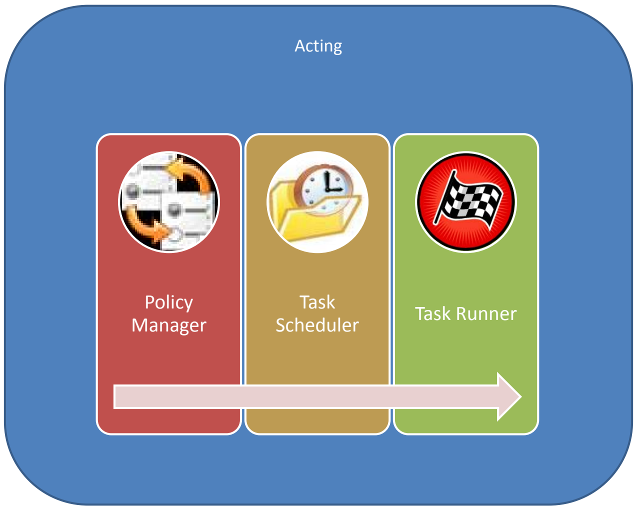


Figure 5. Tasks of the acting process.

A. Policy Manager

The decision of carrying out a task must be supervised by the Policy Manager. A smart environment may have defined policies about energy saving, security or comfort. This manager may implement the software artifacts necessary to define, store and query these policies such as Rei policy language [24] or a more extensive work of [25].

This process must be also suitable to change its policies in real time, so inhabitants preferences may vary (i.e. during vacation periods energy saving and security could be a priority in the smart environment).

B. Task Scheduler

Once an action has been ordered by the reasoning process, and has passed through the filter of the policy manager it has to be scheduled.

The scheduler has to consider that some tasks could have a time limit to be executed, and also has to consider its priority. The implementation will typically implement a queue where action will wait.

C. Task Runner

The task runner is the last task of the *Acting* process. It has to deal with low level protocols to send orders to devices. This task will receive a set of orders for a set of devices and it will have to translate them to low-level instructions.

This task must be a very light process with almost no-lag so its time xcomplexity should be lineal.

VI. Case study

We have already shown the Reference Architecture proposed, and the developing environment in [21] so it's time to present the applications developed during this work to validate the Architecture.

We have focused on developing an implementation of the Perception process (first of the Reference Architecture) which main objective is the *perception* of data about localization of workers, luminosity, temperature and humidity of an office at the LSI department at the University of Seville.

A. Mote Sensor Report

This application belongs to the *Data Collector* task. As it was presented in section III-A this task has the responsibility of the low-level interaction with devices, and it could be distributed between central stations of the Smart Environment and the devices themselves. Mote Sensor Report is a pervasive application developed on Sentilla Work IDE and deployed over Sentilla Tmotes.

Sentilla Tmotes implement a JVM called Sentilla Point so it can run Java applications. In order to access the hardware capabilities of the device, Sentilla offers a java library which give you access to the data gathered by the sensors and other elements such us extension port or leds.

The java application developed access to sensor data and send it through the Zigbee interface. The sensors conform a mesh network where motes act as repeaters. This type of network makes possible to cover wider areas even though Zigbee protocol has a 10m. radio.

When developing this application we realized that luminosity captured by sensors fluctuated if the fluorescent lights of the office were on. The reason of this behavior is that fluorescent bulbs are always turning off and on although it is not noticeable by humans due to its short frequency. However this behavior was a problem, so we had to tackle it and we included a part of the *Repairer* task (see section III-C) at this point, due to the reparation couldn't be possible in a later moment so it has to be tackled at this point.

The solution is quite simple, instead of retrieving just one value, n values are retrieved and the average of them is calculated and then sent to the Sentilla Gateway plugged into the central station.

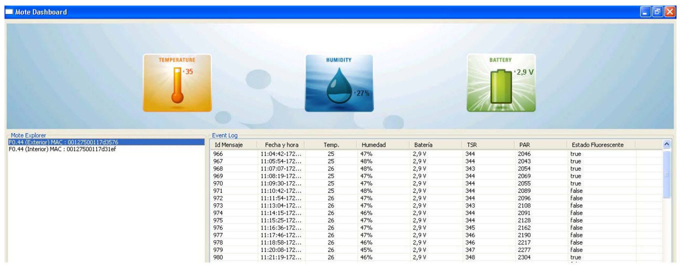
B. Mote Dashboard

In order to receive all the information from the Sentilla *motes* an application has been developed. It shows the information that is being received from the motes in real time. This application carries out three main tasks:

- **Data Collector**. An execution thread is responsible of managing the reception of data from the motes, using the mote gateway supplied with the Development kit.
- **Verifier**. A simple verifier has been developed and performs simple tests, similar to preconditions, over data received. (e.g. $luminosity >= 0$).
- **Repairer**. When the verifier task detects erroneous data Mote Dashboard choose between two options:
  - Replace. If previous correct data was received short time before, concrete erroneous values are replaced with previous values. The time difference is customizable.
  - Reject. On the other hand, if previous correct data was received long time before, current erroneous data is rejected.
- **Ontologiser**. Finally correct data is processed by the ontologiser task. It performs three operations:
  - Synchronization. It keeps a buffer of data received from every *mote*. When information from outdoor and indoor *motes* was taken at approximately same time it is composed by the aggregation task.
  - Aggregation. Once data is synchronized, it is added to the same instance of the input data.
  - Store. Finally, Dashboard stores all instances in *arff* format and *SQL* database (useful for machine learning tools like Weka [13]).

Mote Dashboard has other minor functionalities:

- **Show information**. It shows the information received from the motes in a GUI (Fig. VI-B), and a list of active motes in the mesh network.
- **Reset**. Resets the dashboard GUI. This doesn't affect stored data, but cleans buffers and GUI from data received.



Mote Explorer

F0.44 (Exterior) MAC : 00127500117d3576
F0.44 (Interior) MAC : 00127500117d31ef

Event Log

Id Mensaje	Fecha y hora	Temp.	Humedad	Batería	TSR	PAR	Estado Fluorescente
966	11:04:42-172...	25	47%	2,9 V	344	2046	true
967	11:05:54-172...	25	48%	2,9 V	344	2043	true
968	11:07:07-172...	26	48%	2,9 V	343	2054	true
969	11:08:19-172...	25	47%	2,9 V	344	2069	true
970	11:09:30-172...	25	47%	2,9 V	344	2055	true
971	11:10:42-172...	25	48%	2,9 V	344	2089	false
972	11:11:54-172...	26	47%	2,9 V	344	2096	false
973	11:13:04-172...	26	47%	2,9 V	343	2108	false
974	11:14:15-172...	26	46%	2,9 V	344	2091	false
975	11:15:25-172...	26	47%	2,9 V	344	2128	false
976	11:16:36-172...	26	47%	2,9 V	345	2162	false
977	11:17:46-172...	26	47%	2,9 V	346	2190	false
978	11:18:58-172...	26	46%	2,9 V	346	2217	false
979	11:20:08-172...	26	45%	2,9 V	347	2277	false
980	11:21:19-172...	26	46%	2,9 V	348	2304	true

Figure 6. Mote Dashboard GUI.

VII. Conclusions

We have shown a General Software Reference Architecture to develop Smart Environment Applications which is very useful for developers of Smart Environments solutions. We have answered questions like: *a*) How should the applications/software be divided?, *b*) Which process should be present for these smart solutions?, *c*) How these process should interact? .

Future work has to focus on continue developing smart applications to provide more feedback to the Reference Architecture in order to continue improving it. Another interesting challenge is to consider not only the ability of the smart environment to fit user preferences but using Smart Environment to change behaviors in the individual. The increasingly pressure requirement of sustainability points out that future smart environments will *teach* the inhabitants to be more conscious with the world and its environment and we believe that it has to be the way future smart environments must work.

Acknowledgment

This research is partially supported by the MEC I+D project InCare. Ref: TSI2006-13390-C02-02 and the Andalusian Excellence I+D project CUBICO Ref: TIC2141.

References

[1] G. Youngblood, E. Heierman, L. Holder, and D. Cook, "Automation intelligence for the smart environment," *Proceedings of the International Joint Conference on Artificial Intelligence*, 2005. [Online]. Available: http://ijcai.org/papers/post-0192.pdf

[2] M. Weiser, "The computer for the 21st century," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 3, no. 3, pp. 3–11, 1999. [Online]. Available: http://www.ece.ubc.ca/~guenther/LearningInMulti-AgentControlOfSmartMatter.pdf

[3] J. A. Alvarez, M. D. Cruz, A. Fernndez, J. A. Ortega, and J. Torres, "Experiencias en entornos de computacin ubicua mediante arquitecturas orientadas a servicios," *CEDI-JSWeb*, pp. 167–174, sept 2005.

[4] D. Cook and S. Das, "How smart are our environments? An updated look at the state of the art," *Pervasive and Mobile Computing*, vol. 3, no. 2, pp. 53–73, 2007. [Online]. Available: http://www.sciencedirect.com/science?_ob=ArticleURL&_udi=B7MF1-4MP002G-1&_user=603129&_rdoc=1&_fmt=&_orig=search&_sort=d&view=c&_acct=C000031118&_version=1&_urlVersion=0&_userid=603129&md5=560f54d55b3fe9bdeb9c64f72b45e751

[5] C. da Costa, A. Yamin, and C. Geyer, "Toward a general software infrastructure for ubiquitous computing," *Pervasive Computing, IEEE*, vol. 7, no. 1, pp. 64–73, Jan.-March 2008.

[6] T. Weis, M. Knoll, A. Ulbrich, G. Muhl, and A. Brandle, "Rapid prototyping for pervasive applications," *Pervasive Computing, IEEE*, vol. 6, no. 2, pp. 76–84, April-June 2007.

[7] K. Rehman, F. Stajano, and G. Coulouris, "An architecture for interactive context-aware applications," *Pervasive Computing, IEEE*, vol. 6, no. 1, pp. 73–80, Jan.-March 2007.

[8] D. Bannach, P. Lukowicz, and O. Amft, “Rapid prototyping of activity recognition applications,” *Pervasive Computing, IEEE*, vol. 7, no. 2, pp. 22–31, April-June 2008.

[9] A. Fernández-Montes, J. Álvarez, J. Ortega, M. Cruz, L. González, and F. Velasco, “Modeling Smart Homes for Prediction Algorithms,” *Lecture Notes in Computer Science*, vol. 4693, p. 26, 2007. [Online]. Available: http://www.springerlink.com/content/m6q1051w442nv845/fulltext.pdf

[10] D. Cook, M. Youngblood, and S. Das, “A multi-agent approach to controlling a smart environment,” *Lecture notes in computer science*, vol. 4008, p. 165, 2006. [Online]. Available: http://www.springerlink.com/content/142027w275531607/fulltext.pdf

[11] S. Das and D. Cook, “Designing Smart Environments: A Paradigm Based on Learning and Prediction,” *Mobile, Wireless, and Sensor Networks*. [Online]. Available: http://www.springerlink.com/content/7t87564r42285713/fulltext.pdf

[12] J. Li, Y. Bu, S. Chen, X. Tao, and J. Lu, “FollowMe: On Research of Pluggable Infrastructure for Context-Awareness,” *Proceedings of the 20th International Conference on Advanced Information Networking and Applications-Volume 1 (AINA’06)-Volume 01*, pp. 199–204, 2006. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/AINA.2006.182

[13] I. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2005.

[14] F. Doctor, H. Hagras, and V. Callaghan, “A fuzzy embedded agent-based approach for realizing ambient intelligence in intelligent inhabited environments,” *Systems, Man and Cybernetics, Part A, IEEE Transactions on*, vol. 35, no. 1, pp. 55–65, Jan. 2005.

[15] J. Augusto and C. Nugent, “The use of temporal reasoning and management of complex events in smart homes,” *Proceedings of European Conference on Artificial Intelligence (ECAI 2004)*, 2004. [Online]. Available: http://www.infj.ulst.ac.uk/~jcaug/ecai2004.pdf

[16] C. Bishop, *Neural Networks for Pattern Recognition*. Oxford University Press, USA, 1995.

[17] D. MacKay, *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003. [Online]. Available: http://www.inference.phy.cam.ac.uk/itprnn/book.pdf

[18] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines*. Cambridge University Press, 2000. [Online]. Available: http://books.google.es/books?hl=es&lr=&id=L-2Vqx56J5UC&oi=fnd&pg=PR9&dq=an+introduction+to+support+vector+machines+cristianini&ots=eszbtwZefz&sig=TiSbqVWHmNNZ-WS-OmReCn9soZI

[19] M. Mozer, “Lessons from an adaptive home,” *Smart Environments: Technology, Protocols, and Applications*, pp. 273–298, 2005. [Online]. Available: http://www.cs.colorado.edu/~mozer/papers/reprints/smart_environments.pdf

[20] K. Larson, “House_n Living Laboratory White Paper,” 2001.

[21] A. Fernandez-Montes, J. Ortega, L. Gonzalez, J. Alvarez, and M. Cruz, “Smart Environment Vectorization: An Approach to Learning of User Lighting Preferences,” *Lecture Notes in Computer Science*, vol. 5177, pp. 765–772, 2008.

[22] H. Hagras, V. Callaghan, M. Colley, G. Clarke, A. Pounds-Cornish, and H. Duman, “Creating an Ambient-Intelligence Environment Using Embedded Agents,” 2004. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/MIS.2004.61

[23] N. Roy, A. Roy, and S. Das, “Context-Aware Resource Management in Multi-Inhabitant Smart Homes: A Nash H-Learning based Approach,” *Proc. of 4th IEEE Int’l Conf. on Pervasive Computing and Communications (PerCom2006)*, 2006. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/PERCOM.2006.18

[24] L. Kagal, T. Finin, and A. Joshi, “A policy language for a pervasive computing environment,” *Policies for Distributed Systems and Networks, 2003. Proceedings. POLICY 2003. IEEE 4th International Workshop on*, pp. 63–74, June 2003. [Online]. Available: http://ieeexplore.ieee.org/iel5/8577/27164/01206958.pdf?tp=&isnumber=&arnumber=1206958

[25] J. Ahn, B. Chang, and K. Doh, “A Policy Description Language for Context-based Access Control and Adaptation in Ubiquitous Environment,” *TRUST06, August*, 2006. [Online]. Available: http://www.springerlink.com/content/67g537107314j573/fulltext.pdf