An Ontology-Based Autonomic System for Ambient Intelligence Scenarios

Alessandra De Paola

Abstract Pervasive computing and Ambient Intelligence (AmI) demonstrate that computer systems which directly interact with users are characterized by increasing size and complexity, so that the human user will still not be able to adequately manage them for a long time to come. As a response to this trend, the Autonomic Computing paradigm aims to design and develop systems able to self-configure and self-manage. The research reported here is part of an AmI project that proposes a multi-tier cognitive architecture for aggregating sensory information at different levels of abstraction. In such an architecture, a central reasoning component is able to understand the environmental state and the user's preferences and consequently to plan the opportune actions to be performed. This chapter describes an ontology able to provide a formal representation of the environment in which the AmI system is placed, as well as a representation of the system itself and of its interaction with the environment. By exploiting this knowledge, the AmI system can develop consciousness of itself and of its cognitive processes, and consequently the capability of autonomously managing its own functioning. In particular, this task is performed by a rule-based planning module, integrated within the multi-level architecture, and capable of managing and configuring the sensory infrastructure. By means of this module, the AmI system can manage its own monitoring activity to obtain a good understanding of the context while minimizing system energy consumption.

1 Introduction

Ambient Intelligence (AmI) [2, 8] is based on the integration of the Internet of Things (IoT) [1] and Artificial Intelligence. This paradigm defines an application scenario where the user is the focus of a pervasive environment augmented with sensors and

A. De Paola (✉)
University of Palermo, Viale delle Scienze, ed 6, 90128 Palermo, Italy
e-mail: alessandra.depaola@unipa.it

S. Gaglio and G. Lo Re (eds.), *Advances onto the Internet of Things*,
Advances in Intelligent Systems and Computing 260, DOI: 10.1007/978-3-319-03992-3_1,

actuators, enabling an intelligent system to monitor the environmental conditions and to perform actions aimed at satisfying user requirements [25].

Although the main goal of AmI is not the development of pervasive sensory and actuator devices, their availability is necessary to develop AmI systems, because such devices enable the development of a pervasive sensory infrastructure, while maintaining a low degree of intrusiveness, and with low costs of production, deployment and maintenance.

The adoption of such sensory infrastructure poses new challenges, involving both the information management process and the analysis of data gathered to detect anomalous behavior [9]. In order to take full advantage of the raw information gathered by pervasive devices, information has to be properly represented to extract meaningful knowledge. For this purpose, the work reported here adopts a multi-level cognitive architecture [5], whose sensory infrastructure is based on Wireless Sensor Networks (WSN) [29, 31]. Sensor nodes gather environmental data and forward them towards a central intelligent engine, where high-level reasoning occurs. Sensory information is then aggregated and processed inside the reasoning engine by a stack of modules, adopting the most appropriate representation according to the level of abstraction at which information is processed. The knowledge achieved in this way is exploited to perform high-level inferences about the perceived context and about user's preferences and needs, in order to plan the most suitable actions to be performed to meet system goals.

Within such a scenario, this work proposes an ontology capable of supporting the design and the development of AmI systems based on the adopted multi-tier architecture. A methodology for developing autonomic behavior to self-manage the monitoring system is also proposed. The ontology described here provides a formal representation both of the specific application domain and of the AmI system itself. In addition to driving the process of knowledge abstraction, from raw sensory data up to higher-level concepts, it enables the AmI system to gain consciousness of itself, of its own interactions with the environment and of its own cognitive activity, thus enabling the autonomic management of its own behavior.

An ad-hoc module exploits the system's ontological representation to self-configure its monitoring activities. In particular, by means of an introspective analysis of its own state and of its cognitive activity, the system is able to define symbolic plans to be translated into commands to be given to the actuators, to self-configure the sensory infrastructure.

The case study selected to verify the potential of the proposed approach is a Building Management System (BMS) for controlling ambient conditions of an office environment, in terms of heating, ventilation, air conditioning (HVAC) and lighting, with the goal of maximizing user comfort while reducing energy consumption of the sensory infrastructure.

The chapter is organized into six sections. Section 2 introduces AmI and Autonomic Computing paradigms, highlighting the ways they relate to each other, and briefly describing some relevant approaches proposed in the literature. Section 3 describes the main features of multi-tier architecture adopted, specifying the role of the proposed ontology and introducing self-management features. Section 4 explains the

proposed ontology and examines some architectural details of the system. Section 5 describes the component devoted to the dynamic self-configuration of the sensory infrastructure, exploiting a rule-based reasoning engine. Finally, Sect. 6 sets out the conclusions drawn by the author and outlines possible future developments.

2 Ambient Intelligence and Autonomic Computing

Ambient Intelligence and Autonomic Computing are two emerging paradigms, which have many features in common. Some relevant desirable features for AmI systems, such as self-management capability, adaptivity to highly dynamic scenarios, context-awareness, monitoring and analysis capability, represent the essence of Autonomic Computing. The development of autonomic systems can therefore be considered a technological prerequisite for designing AmI applications.

Many approaches presented in the AmI literature exploit artificial intelligence (AI) techniques to manage the huge set of devices deployed in the environment, to cope with the intrinsic uncertainty and imprecision of environmental models, and to adapt the system to changes in environmental conditions and user behavior.

Among the most widely adopted AI techniques, are neural networks, fuzzy systems and Bayesian networks [14]. The authors of [7] propose a non-supervised learning method for a fuzzy system devoted to the control of environment actuators, such as artificial lighting, windows and HVAC systems. The system is able to monitor environmental quantities such as light and temperature, besides the user interactions with actuators. In their Neural Network House, the authors of [22] propose the use of neural networks, together with rules for occupancy detection, to predict the binary occupational state of monitored sites. Input data for the neural network is provided by sensory readings from binary motion sensors. The authors of [20] use a Bayesian network to identify the sequence of actions to be performed on the actuators in order to carefully imitate a user's past behavior.

Independently of the approaches adopted, an AmI system has to include knowledge of itself and the environment in which it acts. Some works [13, 19, 24] have focused on the forms of representation and communication of this knowledge, and on the semantic enrichment of data processed by the system through the use of ontologies. This self-consciousness enables complex systems to autonomously perform configuration, maintenance, management and optimization, which are all tasks typically assigned to human operators. This development trend is desirable because AmI systems are becoming increasingly complex, distributed and heterogeneous, and consequently designers are no longer able to predict in advance all possible patterns of interaction between components [16]. This philosophy is driven by the initiative launched by IBM in 2001 [18], which called it "Autonomic Computing", in reference to the human autonomic nervous system.

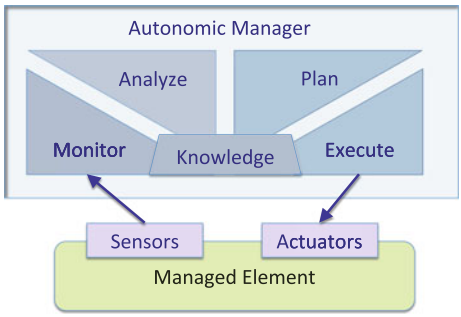

Fig. 1 The monitor-analyze-plan-execute cycle of autonomic computing [32]

The main Autonomic Computing functionalities are listed below:

- *self-configuration*: the system is able to self-configure according to high-level policies defined by human operators, autonomously composing new diverse subsystems;
- *self-optimization*: the system is able to tune its own parameters in order to optimize its own behaviour;
- *self-healing*: the system is able to identify problems and causes of failure, and to solve them , thus returning the system to normal functioning;
- *self-protection*: the system is able to protect itself against malicious attacks and to anticipate possible failures.

In order to enable those functionalities, a system needs a sensory infrastructure to monitor its behavior, analysis modules to detect relevant events from sensory data, and reasoning modules to plan the sequence of actions to be performed using a set of actuators [15, 16, 32]. The resulting monitor-analyze-plan-execute cycle is showed in Fig. 1.

The features and goals of Autonomic Computing grow in importance as Ambient Intelligence becomes more pervasive and dynamic. In such a scenario, in order to make the AmI system invisible to the user, the system needs to be capable of autonomously interacting with the environment, of self-managing only on the basis of high-level policy, and of dynamically learning user preferences.

It is thus evident how autonomic capabilities are a basic requirement for making future AmI systems capable of effectively coping with diverse contexts, without become a burden for human operators and users. In other words, AmI represents a vision of a future world in which the IoT and the Artificial Intelligence cooperate in a scenario which requires complex features, such as dynamism, adaptability, non-intrusiveness and self-management, all facilitated by the Autonomic Computing paradigm, which thus becomes a necessary condition for its realization.

3 A Multi-Level Approach for AmI Applications

This chapter proposes the adoption of the cognitive multi-tier architecture for AmI systems described in [4, 5]. Such an architecture adopts a distributed sensory infrastructure for gathering environmental and context information, and different levels of abstraction for representing and processing knowledge. In such a framework, intelligent modules make it possible to understand user needs, to plan the sequence of actions to be performed, and to self-configure system behavior.

The architecture adopted consists of four subsystems, each of them characterized by a modular and configurable structure, as defined below:

- *Sensing subsystem*: based on Wireless Sensor Networks (WSNs), it perceives relevant ambient information, and sends raw data toward the understanding subsystem;
- *Understanding subsystem*: it processes data through different levels of abstraction to extract high-level knowledge; at the highest level it provides a concise and meaningful description of the current context;
- *Planning subsystem*: it exploits the ambient description provided by the understanding subsystem to plan the most appropriate sequence of actions to be performed to satisfy user needs and optimize the system's behavior;
- *Actuation subsystem*: consisting of all actuators able to modify the environment status, i.e., heating, ventilation, air conditioning (HVAC) and lighting systems; available actuators receive control commands from the planning subsystem.

The understanding subsystem is split into multiple levels, according to a multi-tier structure of interconnected modules for representing knowledge. In particular, there are three types of module, namely "subsymbolic", "conceptual", and "symbolic" modules. Knowledge flows through those tiers, assuming the suitable form required by each module. A more detailed view of the system structure and of the role played by each component, is provided in Sect. 4 together with a description of the ontology proposed here.

In the case study considered here, the *sensing subsystem* is composed of a WSN, whose nodes are deployed in different locations inside the controlled premises, for monitoring relevant physical quantities, namely, temperature, humidity and lighting level. Moreover, a set of sensors on actuators make it possible to monitor user activity and obtain implicit feedback relating to comfort levels. The sensory data gathered are processed by the *understanding subsystem*, which builds a concise representation of current environmental condition and context, such as information about user presence or activity, about current user comfort, and about energy consumption. To take one example, let's consider a set of sensors able to catch the switching on/off of the artificial lighting system. This raw information allows the system (i) to infer the current status of the artificial lighting system and to correlate it to the corresponding lighting level, (ii) to learn the correspondence between the actuator status and its current energy consumption, and (iii) to infer the appreciation level of users about the current lighting conditions.

4 The Proposed Ontology

As highlighted by a considerable body of research in the literature, an AmI system needs to acquire, implicitly or explicitly, the knowledge of the scenario being considered and of its own interaction with it [13, 19, 24]. Moreover, if the autonomic capability of self-management is required, the AmI system needs to have an internal representation of its own structure and of interactions between its own modules [16, 27, 30, 32]. A formal and explicit representation of such knowledge can be obtained by defining an ontology, which represents an unambiguous vocabulary consisting of the definitions of classes in the domain of interest and of relations among them [12, 26].

The ontology proposed here serves several purposes:

- to provide an unique definition of the concepts on which the system is based, to support the design phase;
- to make it possible to define a generic framework, easily configurable for different application of the AmI system;
- to allow the system administrator to express only high-level goals and neglect low-level details;
- to facilitate automatic interaction between system modules able to describe their own services;
- to support the development of the autonomic capabilities of the systems.

Thus, the ontology defines the environment and its properties, the user and his interaction with it, as well as system components, their interconnections and their relationships with the environment. For instance, it is possible to represent how data flows within the system, or what relationship exists between sensory devices and the environmental properties perceived by them.

The representation employed here can be sub-divided into two ontologies: the *General Ontology* and the *Domain Ontology*. The former represents the structure of a generic system and its possible relationships with the environment. It is not tied to a specific application scenario, and does not therefore have a specific configuration of system modules. The *Domain Ontology*, on the other hand, includes knowledge about an instance of the system for a specific scenario, in our case the BMS for environmental comfort in an office.

Both ontologies are described by the OWL DL language, and the domain ontology is anchored to general one through the import mechanism allowed by the OWL [21]. The choice of OWL DL allows the designer to exploit the services of automatic control of consistency, of classification and inference, and of verification of the correctness of the coded knowledge.

A set of logical rules related to the proposed ontology makes it possible to infer new knowledge directly by processing the ontology. These rules are expressed by means of the Semantic Web Rule Language (SWRL) [17], as new OWL axioms, with the classic form `antecedents` $\rightarrow$ `consequent`. SWRL rules can be used to define a property as a composition of other properties, thus expressing the idea that

a set of basic properties imply a composite property. They can also be used to transfer the value of a given property from certain individuals to another related individual.

The use of a rule engine capable of processing ontologies and their logical rules, makes it possible to perform automated reasoning on the domain. A tool for this purpose is the inference engine Jess [10] integrated with the development environment Protégé [11] used to define the ontology. This inference engine, integrated into the symbolic planning module, allows the system to reason both about the environment and the system itself, and then to take the appropriate actions.

4.1 The General Ontology

The *General Ontology* defines classes and properties required to describe the components of the multi-tier architectures, their interconnections, and the data flow inside the system. Moreover, it describes the basic elements constituting the sensory and actuator subsystem.

4.1.1 Environmental Properties and Physical Devices

What the AmI system is able to monitor is defined as an `AmbientProperty`; this class is further specialized into three separate subclasses: `PhysicalProperty`, representing physical observable phenomena such as temperature and humidity, `Status`, representing the status of a particular environmental element, such as a door in the `CLOSED` state, and `Event`, representing observable events such as a device fault or the entry of the user into a monitored room.

Physical devices controllable by the AmI system, and composing the sensory and actuator infrastructure, are modeled by means of the `Device` class. This class is subdivided into three specialized subclasses, namely `Sensor`, `Actuator`, and `Node`. Each device is deployed in a specific room, and this topological relationship is also represented inside the ontology.

The Planning subsystem controls a generic device by sending an opportune command. This dependency is expressed inside the ontology which also models the data flow inside the system by means of a set of properties. In particular, the property `hasInputFrom` makes it possible to code the fact that a `Device` is able to receive input only from an `ActuatorModule`, that is a specific module of the multi-tier architecture. Analogously, the ontology models the fact that a `Sensor` is able to produce input data for a specific type of module, namely for a `SubsymbolicModule`.

Besides these properties, the ontology models other characteristics of a sensor, such as sampling rate, continuity of monitoring, energy consumption, and the node over which the sensor is installed. Similarly, each actuator is also characterized by the set of commands that it is able to receive.

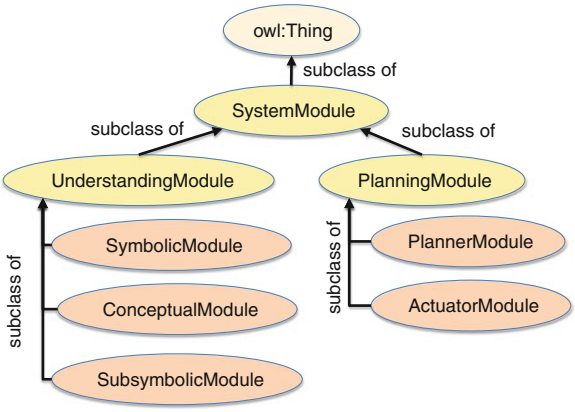

Fig. 2 Taxonomy of the `SystemModule` class in the proposed ontology. Graph nodes represent ontology classes, and *arrows* represent properties, going from a class of the property domain toward a class of the property co-domain

An important relationship is the connection between an ambient property and sensors able to perceive it. This relation is exploited by a SWRL rule to determine which properties are observable or not:

$$\texttt{AmbientProperty(?x)} \wedge \texttt{Sensor(?y)} \wedge \texttt{senses(?y, ?x)} \rightarrow \texttt{ObservableProperty(?x)}$$

A similar but specular relationship exists between actuators and ambient properties. i.e., `actsOn`, making it possible to model which actuators are able to modify a particular environmental condition.

4.1.2 Architecture Modules

Architecture modules are modeled inside the ontology by means of the `SystemModule` class, whose descending taxonomy is shown in Fig. 2. The main distinction is functional, distinguishing between `UnderstandingModule` and `PlanningModule`, as described in Sect. 3.

The interconnections between modules are coded by means of the `hasInputFrom` and `hasOutputTo` properties. The knowledge representation for each module, in terms of coding for input and output, is described by means of the `hasInputData` and `hasOutputData` properties, with instances of `DataType` classes as their dominion.

The `UnderstandingModule` is further specialized in the following classes: `SubsymbolicModule`, `ConceptualModule`, and `SymbolicModule`. The organization into levels of increasing abstraction is constrained by the `hasOutputTo` property, which can take values from the `SubsymbolicModule` class for the `Sensor` and `Node` classes, from the `ConceptualModule` class for

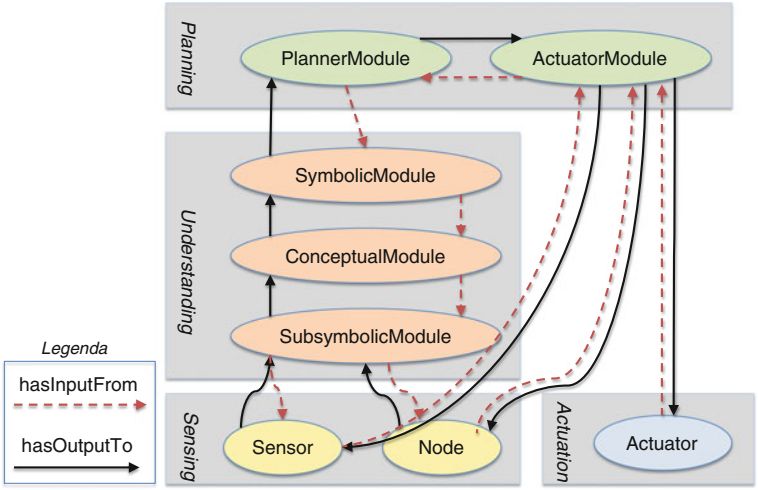

Fig. 3 Three-tier organization of the understanding subsystem, as coded by the proposed ontology

the `SubsymbolicModule` class, and from the `SymbolicModule` class for the `ConceptualModule` class. Data produced by a `SymbolicModule` are directly usable by a `PlannerModule`. This type of module acts at the highest level of abstraction, with the aim of reasoning on symbolic representations of environment and context, and thus producing high level plans designed to achieve system goals. Plans are processed by `ActuatorModules`, which translate them into low-level commands for `Actuators`, to modify environmental conditions, and for `Nodes` and `Sensors`, to modify the behavior of the sensing subsystem. This interconnection pattern is shown in Fig. 3.

4.2 The Domain Ontology

The *domain ontology* imports and extends the *general ontology*, by defining subclasses and individuals to describe a specific instance of the AmI system. The application scenario considered in this chapter is *Sensor9k* [6], a testbed, designed for an office environment, and for reasoning about user comfort and energy saving. Different nodes are equipped with several types of sensors capable of monitoring the environment. The understanding subsystem is capable of estimating the number of users in the monitored office, and evaluating the current lighting level. This information is then combined to evaluate the adequateness of the lighting level. A simple planning module uses such high level evaluation to determine which action needs to be performed in order to achieve an adequate lighting level with minimum energy consumption.

4.2.1 Environmental Properties and Physical Devices

In the domain under consideration, the `PhysicalProperty` class has the following instances: `Light`, `Sound`, `Pressure`, `Temperature`, `Humidity`. The `Event` class has the following instances: `Activity`, representing the occurrence of an interaction between the user and some of the manual actuators deployed in the environment, such as a light switch; `RFIDPassing`, representing the proximity of an RFID tag to an RFID reader (this event is correlated with the presence of the user in a monitored room); `WorkstationActivity`, representing the interaction of the user with his/her workstation. The `Status` class has the following instances: `DoorStatus`, representing the closed / open / locked status of a door; `UserInOffice`, representing the presence of the user in his/her own office; `RoomOccupancy`, representing the number of people in a monitored room; `UserInBuilding`, representing the presence of the user in the building being monitored.

The domain ontology obviously contains a set of individuals for topological classes, and the correct values for properties that make it possible to specify the placement of each device. Moreover, each sensor or actuator is linked with the ambient property that it is able to monitor or modify.

4.2.2 Architecture Modules

The software architecture is specified by means of the definition of a set of subclasses and individuals of the `SystemModule` class. Each class inherited from the *general ontology* is further divided into specialized sub-classes for modules capable of performing a specific type of reasoning. The taxonomic organization of `SystemModules` in the *general ontology* reflects the level of abstraction at which the knowledge is processed, whilst the further decomposition coded in the *domain ontology* reflects the topological and semantic organization of the monitored environment.

Figure 4 shows an example of such organization. Suppose that the target building consists of two rooms, `room_1` and `room_2`, and that each of these rooms contains light sensors. The software architecture includes a class of subsymbolic modules devoted to processing lighting information, called `Light_ssM`, and two instances of this class, namely `Light_ssM_1` and `Light_ssM_2`. These two modules require as input the same type of sensory data, i.e., `LightReading`, and produce as output the same type of qualitative data, `LightLevel`.

5 Autonomic Self-Configuration

The AmI system proposed here is based on the *monitor-analyze-plan-execute* cycle of a typical autonomous system. This paradigm is exploited not only in controlling the environment surrounding the user, but also in dynamically controlling the behavior of the system itself.

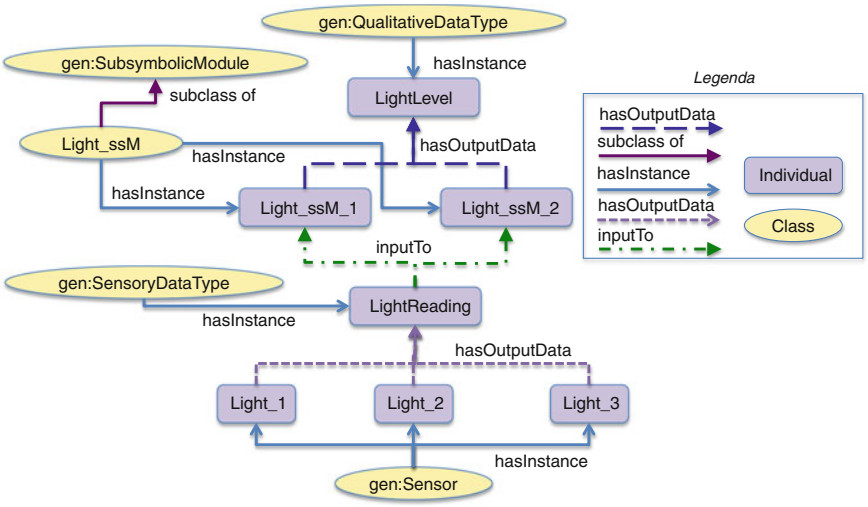

Fig. 4 Example of an AmI system that includes two instances of the `Light_ssM` class; these modules accept as input the same instance of `SensoryDataType` even though the sensory readings come from different sensors; these modules provide, as output, data belonging to the same instance of `QualitativeDataType` class

The application described above for controlling the lighting system is an example of the implementation of such a paradigm for environmental management. The system monitors the environment by gathering sensory readings, analyzes them to obtain a high-level representation of the overall lighting level, plans the sequences of actions to be performed in order to achieve an adequate lighting level, and then executes those actions by means of the actuators available to it.

The implementation of the *Autonomous Computing* paradigm for controlling the system's behavior is embodied in the dynamic configuration of the monitoring activity. The system is able to receive a set of requirements and high-level policies, such as the goal of minimizing the energy consumption of the sensing infrastructure when users are not present in target premises.

Thanks to its capacity for introspection, the system evaluates the quality of its own analysis functionality, and knows which hardware and software elements contribute to inferring a given concept. A planning module, called `SensingPlanner`, exploits this knowledge to optimize the monitoring functionality of the system; this module is based on a set of rules that embodies high-level policies for system management. Information about the system state and inference accuracy flow through different levels of the multi-tier architecture.

In particular, data about sensors' energy consumption are classified by opportune conceptual modules which state whether the consumption is `high`, `medium` or `low`. These linguistic labels are then represented in a symbolic form, by means of the assertion of facts inside the rule engine. Symbolic plans produced by the planning module are provided as input to actuation modules able to translate them into a set of

configuration commands for the sensors. Information about the node states is only relevant for devices fed by batteries, and include some indication of residual energy. Information about the sensor state includes the `on`/`off` state, and the sampling rate adopted. The sampling rate is classified by an opportune conceptual module, that gives it a linguistic label in the following set: {`min`, `low`, `medium`, `high`, `max`}.

The SensingPlanner exploits these facts and its rules to produce commands for sensors (e.g., request to decrease or increase the sampling rate, switching on/off requests), or even alerts for the system administrator.

5.1 Rule-Based Reasoning

The SensingPlanner bases its reasoning on the Jess rule engine. Java Expert System Shell (Jess) is a rule-based environment, that makes it possible to define logical rules in a LISP-like syntax. A Jess application consists of a working memory, a set of rules and an inference engine responsible for applying rules to the working memory.

The working memory consists of a set of facts, which are true statements about the dominion under consideration, and represents the system's knowledge of the world. Facts can be asserted to and retracted from working memory. Each fact has a template (the relation between facts and templates corresponds to the relation between objects and classes in the OOP).

Rules react to changes in working memory, and can exploit auxiliary functions and queries. Each rule is activated when all its antecedents are satisfied, that is when the working memory contains facts matching rule antecedents. Activated rules are fired when the inference engine is executed, thus causing the function execution. Each rule is executed once for any given set of facts in the working memory; a new execution requires that new facts have been asserted.

The templates and rules of the proposed SensingPlanner are described below.

The following templates represent static knowledge:

```
(deftemplate room (slot id))
(deftemplate node (slot id) (slot office)(slot batteryPowered))
(deftemplate nodeSensor (slot nodeId)(slot sensorId))
(deftemplate sensor (slot id) (slot ambientProperty)
(slot energyConsumption) (slot continuousSampling))
(deftemplate user (slot name)(slot office))
(deftemplate ambientProperty (slot id)(slot observable))
(deftemplate affects (slot affectedProperty)
(slot affectingProperty))
```

Facts corresponding to these templates allow the SensingPlanner to know things about the premises being monitored, the users and their offices, and the placement of nodes and sensors. Moreover, these last two templates make it possible to specify which ambient properties can be monitored, and the cause-effect relationship among properties. For example, using these types of facts, the SensingPlanner is able to

know that information about the sound level and the workstation activity provides indirect information about a user's presence.

The following set of templates represents facts about the state of the sensory infrastructure and the context. These facts are dynamically updated on the basis of the current states of devices, of the inferred presence of the user in his office or in the building, and of the accuracy of the monitoring activity.

```
(deftemplate nodeState (slot id)(slot batteryLevel))
(deftemplate sensorState (slot id)(slot sampling))
(deftemplate sensingAccuracy (slot room)(slot ambientProperty)
(slot accuracy))
(deftemplate userInOffice (slot user) (slot present))
(deftemplate userInBuilding (slot user) (slot present))
(deftemplate peopleInRoom (slot room) (slot number))
```

Finally, a set of templates define the structure for facts which represent the output of planning. Plans for tuning sensors, indicating a new value for the sampling rate, {on, off, max, min, up, down}, correspond to the following template:

```
(deftemplate tuneSensor (slot sensorId) (slot newSampling))
```

Alerts for system administrators can be expressed according to the following templates:

```
(deftemplate insufficient-sensing (slot room)
(slot ambientProperty))
(deftemplate short-sensing-life (slot room)
(slot ambientProperty))
```

An `Insufficient-sensing` fact indicates that, although the sampling rate for available sensors is at its maximum sustainable value, the monitoring is not sufficiently accurate and a structural intervention by the system administrator is probably required. A `short-sensing-life` fact indicates that all sensors involved in monitoring a given ambient property in a given room have low residual energy, and so a system administrator intervention is necessary to guarantee that there will be no interruption in the monitoring functionality.

The main rules of the SensingPlanner module are described below.

The first rule states that, for a given room, if all corresponding users are not present in the entire building, then it is possible to switch all sensors off with the exception of those responsible for monitoring the `UserInOffice` property; for these sensors monitoring is minimized.

```
(defrule stopRoomSensing
  (and
    (room (id ?p))
    (not (and (user (name ?u) (office ?p))
              (or (userInBuilding (user ?u)(present "true"))
                (userInOffice (user ?u)(present "true"))))))
  )=>
  (stopRoomSensing ?p)
  (minimizeRoomSensing-ap ?p "UserInOffice"))
```

where minimizeRoomSensing-ap is an auxiliary function which receives as input an ambient property and a room, selects all sensors perceiving this ambient property, directly or indirectly, and sets their sampling rate to the minimum value.

The second rule minimizes monitoring for all ambient properties in a empty room, provided that users normally occupying that room are present in the building.

```
(defrule minimizeRoomSensing
 (and
   (room (id ?p))
   (not (and (user (name ?u) (office ?p))
             (userInOffice (user ?u) (present "true"))))
   (exists (and (user (name ?u) (office ?p))
             (userInBuilding (user ?u)(present "true"))))
 )=>
 (minimizeRoomSensing ?p ))
```

Another two rules deal with increasing and decreasing the sampling rate based on the accuracy of monitoring, inferred by an opportune symbolic module. According to the policy adopted here, an increase in accuracy is considered a goal only if there are people in their offices, so these rules have a lower priority than the previous two. The increaseRoomSensing rule is detailed below.

```
(defrule increaseRoomSensing
 (and
   ?accuracyLowFact <- (sensingAccuracy (room ?p)
               (ambientProperty ?ap) (accuracy "low"))
   (exists (and (user (name ?u)(office ?p))
             (userInOffice (user ?u)(present "true"))))
 )=>
 (if (increaseRoomSensing-ap ?p ?ap)
   then
   (retract ?accuracyLowFact)
   else
   (assert (insufficient-sensing (ambientProperty ?ap)
                          (room ?p)))))
```

This rule increases the monitoring rate of an ambient property characterized by a low level of accuracy, by means of the increaseRoomSensing-ap auxiliary function. This function tries to increase the sampling rate of sensors, starting from sensors with low energy consumption. If none of sensors has any margin for increasing its sampling rate, the increaseRoomSensing-ap function returns the value false, thus causing the assertion of an insufficient-sensing fact, used to trigger a notification to the system administrator.

6 Conclusions and Future Research

This chapter proposes an ontology-based autonomic system for self-configuring the sensory infrastructure of an ambient intelligence system. The ontology makes it possible to define concepts characterizing the environment, the ambient properties upon which the AmI system reasons, and the structure of the system itself.

The cognitive paradigm adopted is characterized by a flexible scheme easily implementable in new scenarios. The system configuration process can take advantage of an ontological representation of system structure and of its interaction with the environment. Within such a predefined and structured framework of basic knowledge, the designer can easily represent the application domain and specify the configuration of system modules, thus reducing the risk of errors.

Moreover, the knowledge of its own structure and of the current context is exploited by a rule-based planner whose goal is to tune the sampling rate of sensors, in order to maximize the accuracy of the reasoning while minimizing the energy consumption of the system.

The system proposed here may be further expanded with the capability of self-instantiating only on the basis of the high-level description provided by the ontology. This outcome would require the development of a library of parametric modules which cover most of the possible types of processing, besides the possibility of extending the library and the ontology with new functionalities.

Another potential future development concerns communication with final users about the cognitive processing which has occurred, and about the reasons that drive the system to take certain decisions [23, 28]. Such a possibility is enabled by the explicit representation of its knowledge.

Finally, it is possible to imagine a cooperative network of intelligent buildings, able to communicate to each other the knowledge about their structure and about what they have learned, in terms of optimal configurations. Naturally, this type of scenario would require integration with a communication protocol able to identify reliable agents [3] with which it would be opportune to cooperate, with the main goal of protecting user privacy.

Acknowledgments This work has been partially supported by the PO FESR 2007/2013 grant G73F11000130004 funding the SmartBuildings project.

References

1. Atzori, L., Iera, A., Morabito, G.: The internet of things: a survey. Comput. Netw. **54**(15), 2787–2805 (2010)
2. Cook, D., Augusto, J., Jakkula, V.: Ambient intelligence: technologies, applications, and opportunities. Pervasive Mobile Comput. **5**(4), 277–298 (2009)
3. Crapanzano, C., Milazzo, F., De Paola, A., Lo Re, G.: Reputation management for distributed service-oriented architectures. In: 2010 Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshop (SASOW), pp. 160–165 (2010)

4. De Paola, A., Farruggia, A., Gaglio, S., Lo Re, G., Ortolani, M.: Exploiting the human factor in a WSN-based system for ambient intelligence. In: International Conference on Complex, Intelligent and Software Intensive Systems, 2009 (CISIS '09), pp. 748–753 (2009)
5. De Paola, A., Gaglio, S., Lo Re, G., Ortolani, M.: An ambient intelligence architecture for extracting knowledge from distributed sensors. In: Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human, pp. 104–109 (2009)
6. De Paola, A., Gaglio, S., Lo Re, G., Ortolani, M.: Sensor9k: a testbed for designing and experimenting with WSN-based ambient intelligence applications. Pervasive Mobile Comput. **8**(3), 448–466 (2012)
7. Doctor, F., Hagras, H., Callaghan, V.: A fuzzy embedded agent-based approach for realizing ambient intelligence in intelligent inhabited environments. IEEE Trans. Syst. Man Cybern. Part A Syst. Hum. **35**(1), 55–65 (2005)
8. Ducatel, K., Bogdanowicz, M., Scapolo, F., Leijten, J., Burgelman, J.C.: Scenarios for ambient intelligence in 2010. Office for Official Publications of the European, Communities (2001)
9. Farruggia, A., Lo Re, G., Ortolani, M.: Probabilistic anomaly detection for wireless sensor networks. In: AI*IA 2011: Artificial Intelligence Around Man and Beyond, Lecture Notes in Computer Science, vol. 6934, pp. 438–444. Springer, Berlin Heidelberg (2011)
10. Friedman, E.: Jess in action: rule-based systems in Java. Manning Publications Co., Greenwich (2003)
11. Gennari, J., Musen, M., Fergerson, R., Grosso, W., Crubézy, M., Eriksson, H., Noy, N., Tu, S.: The evolution of Protégé: an environment for knowledge-based systems development. Int. J. Hum. Comput. Stud. **58**(1), 89–123 (2003)
12. Gruber, T.: A translation approach to portable ontology specifications. Knowl. Acquisition **5**(2), 199–220 (1993)
13. Gu, T., Wang, X., Pung, H., Zhang, D.: An ontology-based context model in intelligent environments. In: Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference, pp. 270–275 (2004)
14. Hagras, H.: Embedding computational intelligence in pervasive spaces. IEEE Pervasive Comput. **6**(3), 85–89 (2007)
15. Hariri, S., Khargharia, B., Chen, H., Yang, J., Zhang, Y., Parashar, M., Liu, H.: The autonomic computing paradigm. Cluster Comput. **9**(1), 5–17 (2006)
16. Herrmann, K., Muhl, G., Geihs, K.: Self management: the solution to complexity or just another problem? IEEE Distrib. Syst. Online **6**(1), 1–17 (2005)
17. Horrocks, I., Patel-Schneider, P., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: a semantic web rule language combining OWL and RuleML. W3C member submission. http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/ (2004)
18. Kephart, J., Chess, D.: The vision of autonomic computing. Computer **36**(1), 41–50 (2003)
19. Klein, M., Schmidt, A., Lauer, R.: 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 (AIM-CU), 30th Annual German Conference on Artificial Intelligence (KI 2007), pp. 1–8 (2007)
20. Kushwaha, N., Kim, M., Kim, D., Cho, W.: An intelligent agent for ubiquitous computing environments: smart home ut-agent. In: Proceedings of the Second IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, pp. 157–159. IEEE Press, Piscataway, NJ, USA (2004)
21. McGuinness, D., Van Harmelen, F.: OWL web ontology language overview. W3C recommendation. http://www.w3.org/TR/owl-features/ (2004)
22. Mozer, M.: The neural network house: an environment hat adapts to its inhabitants. In: Proceedings of the Intelligent Environments AAAI Spring Symposium, pp. 110–114. AAAI, Palo Alto, CA, USA (1998)
23. Pilato, G., Augello, A., Gaglio, S.: Modular knowledge representation in advisor agents for situation awareness. Int. J. Semant. Comput. **5**(1), 33–53 (2011)

24. Preuveneers, D., Bergh, J., Wagelaar, D., Georges, A., Rigole, P., Clerckx, T., Berbers, Y., Coninx, K., Jonckers, V., Bosschere, K.: Towards an extensible context ontology for ambient intelligence. In: Ambient Intelligence, Lecture Notes in Computer Science, vol. 3295, pp. 148–159. Springer, Berlin (2004)
25. Remagnino, P., Foresti, G.: Ambient intelligence: a new multidisciplinary paradigm. IEEE Trans. Syst. Man Cybern. Part A Syst. Hum. **35**(1), 1–6 (2005)
26. Ribino, P., Oliveri, A., Lo Re, G., Gaglio, S.: A knowledge management system based on ontologies. In: International Conference on New Trends in Information and Service Science, 2009. NISS '09, pp. 1025–1033 (2009)
27. Serrano, J., Serrat, J., Strassner, J., O Foghlu, M.: Facilitating autonomic management for service provisioning using ontology-based functions and semantic control. In: IEEE Network Operations and Management Symposium Workshops, 2008, pp. 77–86 (2008)
28. Sorce, S., Augello, A., Santangelo, A., Gentile, A., Genco, A., Gaglio, S., Pilato, G.: Interacting with augmented environments. IEEE Pervasive Comput. **9**(2), 56–58 (2010)
29. Srivastava, M., Culler, D., Estrin, D.: Overview of sensor networks. Computer **37**(8), 41–49 (2004)
30. Stanfel, Z., Hocenski, Z., Martinovic, G.: A self manageable rule driven enterprise application. In: 29th International Conference on Information Technology Interfaces (ITI 2007), pp. 717–722 (2007)
31. Stankovic, J.: Wireless sensor networks. Computer **41**(10), 92–95 (2008)
32. Tesauro, G.: Reinforcement learning in autonomic computing: a manifesto and case studies. IEEE Internet Comput. **11**(1), 22–30 (2007)