An Ambient Assisted Living Framework Supporting Personalization Based on Ontologies

Christophe Jacquet, Ahmed Mohamed
SUPELEC Systems Sciences
Gif-sur-Yvette, France
christophe.jacquet@supelec.fr
ahmed.mohamed@supelec.fr

Marie Mateos, Bruno Jean-Bart
TRIALOG
Paris, France
marie.mateos@trialog.com
bruno.jean-bart@trialog.com

Pierre Bretault, Indrawati Schnepp
ATEGO
Issy-les-Moulineaux, France
pierrebretault@neuf.fr
indrawati.schnepp@atego.com

Yacine Bellik
LIMSI-CNRS
Orsay, France
yacine.bellik@limsi.fr

***Abstract*—As the population in many countries is steadily aging, allowing elderly people to stay longer at home is a growing concern. Ambient Assisted Living (AAL) proposes new techniques to help people remain autonomous, based on ambient intelligence. We present an ontology-based framework in which ontologies enable the expression of users' preferences in order to personalize the system behavior. They are also used for the discovery and interconnection of devices, the storage and retrieval of collected data and the transmission of actions. This way, the behavior of the system may be expressed using high-level logic rules. Another important contribution is the addition of a diagnosis service that monitors the run-time behavior of the AAL system only by using sensors discovered *opportunistically* at run-time and knowledge about physical laws, not pre-defined control loops. Finally, this paper describes an actual implementation, with precise technological details, in order to prove the feasibility of the technical choices, and provide implementation ideas for future projects.**

***Keywords*-Ambient Assisted Living (*AAL*); ambient intelligence; ontologies; diagnosis; reasoning.**

I. Introduction

Due to the demographic change towards an aging population, society must find ways to assist elderly people to stay active at home longer. While currently this support is mainly provided by human caregivers, technology will play a more and more important role both for elderly persons and caregivers. In Europe a roadmap has been defined in the last years called Ambient Assisting Living (AAL) [1]. The business context of AAL is rich in terms of technology (from tele-health systems to robotics) but also in terms of stakeholders (from service providers to policy makers, including core technology or platform developers).

The work presented here has been carried out within the CBDP project (Context-Based Digital Personality) [2], which aims at creating a framework for building various kinds of ambient-intelligent applications, based around the concept of Digital Personality for representing the preferences of users. Aside from AAL, several application domains were considered, such as digital TV guides or assistants for workers at a construction site. Therefore the CBDP framework addresses a wide variety of requirements. In this paper however, we focus exclusively on the parts of the CBDP framework relevant to AAL.

Our approach is entirely based on ontologies. Not only are ontologies used to capture domain knowledge, but more importantly they serve as the runtime mechanism that allows the interconnection of devices, the exchange of data and the execution of actions. Moreover, by examining the sequence of requested actions and observed sensor values found in the ontology, a diagnosis process is able to monitor the run-time behavior of the system and to detect unexpected patterns.

The ontology is presented in Section II. Section III describes the CBDP framework and gives implementation details; Section IV focuses on diagnosis. Section V describes a typical AAL use case, and goes through its complete realization. Section VI introduces some related work, and compares our approach with published results. Finally Section VII gives directions for future work.

II. An Ontology for AAL Applications

CBDP is built around an ontology: this section justifies this choice and describes the ontology used.

A. Why use ontologies?

AAL applications are trans-disciplinary by essence (for instance, they can mix automatic control with modeling of user behavior), therefore the ability to reuse knowledge and integrate several knowledge domains is particularly important for them. Furthermore, the field of AAL is very open and changing, so it is not possible to base an AAL platform on a fixed set of features, on a fixed set of data models: extensibility is key. In addition, an AAL environment may require the interoperation of software and hardware devices from a variety of suppliers: there must be a standard way of exchanging knowledge.

Ontologies are well-adapted to all these needs [3]: an ontology framework provides a standard infrastructure for sharing knowledge. In addition, semantic relationships such as equivalence may be expressed between various knowledge

 ISBN: 978-1-61208-235-6

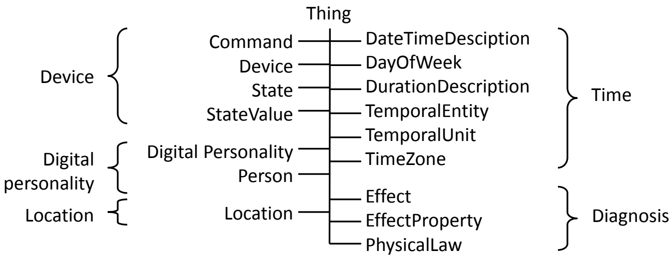

Figure 1. First level of the CBDP ontology.

sources, thus permitting the easy integration of several sources or domains. In addition, one can easily extend an ontology to take into account new applications or new devices. For these reasons, leading AAL projects such as OASIS (Open architecture for Accessible Services Integration and Standardisation) have put a strong emphasis on ontologies [4]. Being oriented toward personalization, CBDP explicitly introduces an ontology module for modeling the “Digital Personality” of the user.

B. Ontology used by CBDP

The ontology defined for CBDP is built around the OWL language [5], which is based on the Resource Description Framework (RDF). RDF represents knowledge as a set of *triples* or *statements* of the form {subject, predicate, object}. It models different interrelated domains in a modular way, so as to enable its easy adaptation to new applications. In order to put into practice the aforementioned notion of reusability, two of the domains are based on existing ontologies. Figure 1 depicts the first level of the ontology; the main domains are as follows:

- *Device:* this part is based on the DogOnt [6] ontology that has been simplified for our purpose, while keeping the modeling axes (typology, functionality and state).
- *Digital Personality:* a class *Person* allows the representation of a human being, and a *Digital Personality* stores the person’s preferences in order to personalize the services offered to him/her.
- *Location:* a location model is required because most of the services offered in the AAL domain must know the position of the user (in/out the house, in the bedroom/in the kitchen, etc.) and of the devices (sensors and actuators).
- *Time:* we import W3C’s existing Time Ontology [7] without any change.
- *Diagnosis:* we introduce the concept of physical effect (see Section IV below), to compute the expected result of the action of an actuator onto a sensor.

The ontology is loosely coupled with the framework, so to a great extent it may be changed without affecting it. However, the basic feature of sending commands to actuators rely on specific *core classes and properties* that may not be changed: this part is depicted on Figure 2.

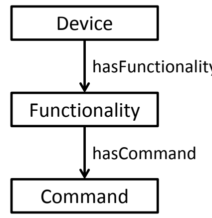

Figure 2. Ontology classes required for proper operation of the framework.

III. CBDP Framework

This section describes the CBDP framework, and how it can be used to build AAL applications.

A. Architecture

The main goal of the CBDP Framework is to dynamically handle ontology data and initiate actions when specified conditions in the ontology are achieved. CBDP is written in Java; it is based on OSGi (Open Services Gateway initiative framework) [8], which allows one to build applications flexibly by combining *bundles*. In CBDP an application is composed of CBDP’s core bundles (the Context Reasoner and the Sensor/Actuator Layer, described in Sections III-B and III-C, respectively) and application-specific bundles (see Figure 3). In our case:

- AAL-specific application bundle: contains the rules that define the intended application behavior, meant to assist the user according to his/her needs.
- Zigbee Driver bundle: allows the exchange of data between the physical devices (connected via a wireless Zigbee network) and the CBDP Framework.

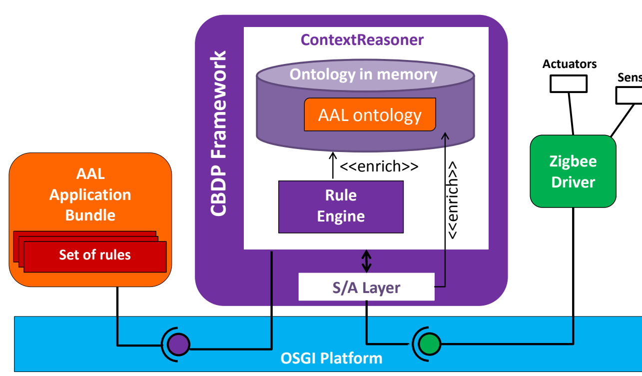

Figure 3. Architecture of the CBDP framework.

B. Context Reasoner and Rules

The Context Reasoner is in charge of managing the information coming from external components (AAL Application or Zigbee Driver) by structuring them according to the AAL ontology. Therefore, it provides methods to add new information, retrieve stored information, and perform queries about that information. Manipulation of the ontology is done using the Jena library [9].

Another feature of the Context Reasoner is its rule engine. Its purpose is to perform actions to help the user and facilitate common tasks, based on a set of application-specific rules (hence the rules are provided by the AAL Application bundle). The rules are Horn clauses [10]: a rule is composed of premises that determine the situations in which the rule applies, and a conclusion, that basically adds a new "fact" into the ontology, such as a new property value. An example of such a rule is given in Section V-A below. Rules are applied by Jena's basic reasoning engine, using forward chaining.

For performance reasons, the rule engine does not apply all rules at each instant. The rules are applied only when a change in the ontology matches a *filter* (i.e., happens in a specific part of the class hierarchy). The filters are application-specific; here they are defined by the AAL Application bundle. At first one may use a "catch-all" filter; performance can be improved later by refining the filters.

C. Sensor/Actuator Layer

The Sensor/Actuator layer (S/A layer) connects the sensors and actuators to the ontology. The communication is two-way:

- Sensor data (sent through Zigbee) is stored in the ontology. This allows one to perform semantic queries and semantic reasoning over sensor data.
- A command request inserted in the ontology (using a property called *hasCommand*) triggers the actual emission of a command to the actuator.

The module responsible for connecting the sensors to the Context Reasoner is based on the use of a specific OSGi service called *EventAdmin*. A communication protocol through OSGi events has been defined in order to allow the communication between the drivers and the S/A layer. Section III-D describes this protocol.

D. Communication between sensors/actuators and the ontology

This section deals with the protocol used to exchange ontology knowledge using OSGi events. An event is composed of a *topic* and of a list of *properties* ({propertyName; propertyValue} pairs). We have defined two kinds of events: 1) to report sensor data, 2) to send commands to actuators. For both kinds of events, the OSGi *topic* string is built according to the pattern `CBDP/AAL/`*`deviceClass`*. `CBDP` and `AAL` are invariant: they reference the general framework and our application-specific ontology; *`deviceClass`* is the name of the sensor class that sends data, or actuator class that is to receive data. The remainder of this section gives details on the actual *contents* (list of properties) of the events in both cases.

Sensor information	
instance.id	URI identifying the sensor (String)
1) When referencing a dataProperty present in the ontology	
data.property	Name of the "simple data" property (String)
data.property.value	Value (depends on property: Boolean, Double, Integer, String...)
2) When referencing an objectProperty present in the ontology	
object.property	Name of the "object" property (String)
object.property.range	Name of the class referenced by the property (String)

Table I
OSGI PROPERTY NAMES USED TO SPECIFY OWL TREES.

1) Reporting sensor data: When sensor data is reported, a sub-graph (actually a tree) must be created in the ontology. An edge in this tree may be of two kinds: connecting an object to a simple value such as an number ("dataProperty"), or connecting an object to another object ("objectProperty"). A convention using OSGi's properties allows us to completely describe the tree. At each node in the tree to be created, a set of datatype and object properties may be specified. Each edge of the tree is numbered using a simple convention: from the top of the tree, each time an edge is followed, a dot and the index of the edge under its parent node are appended to the OSGi property name (see the examples on Figure 4). This permits the description of each edge and each node to be created. The basic property names (without trailing dots) are given in Table I, and a complete example is given on Figure 4. It represents an event stating that the light level is 500 lux in the kitchen at the date {Calendar value}.

2) Sending actuator commands: Sending a command to an actuator is done using the following convention: a new statement must be added in the ontology, with a relation named "hasCommand" (see Figure 2 above). Such a statement may be added by a reasoning rule, or by application code calling the context manager.

When the S/A layer detects a new "hasCommand" statement, it serializes the corresponding sub-tree of the ontology graph into an OSGi event (using the same convention as above) and sends it to the driver of the target actuator.

E. Deployment

The OSGi implementation used by CBDP is Apache Felix. The use of Java and OSGi permits to deploy the framework on a variety of platforms. We have conducted tests on desktop PCs (under Windows and MacOS) and on embedded systems (on a set-top-box running Linux and on Aonix Perc) [11]. Perc is a Java virtual machine

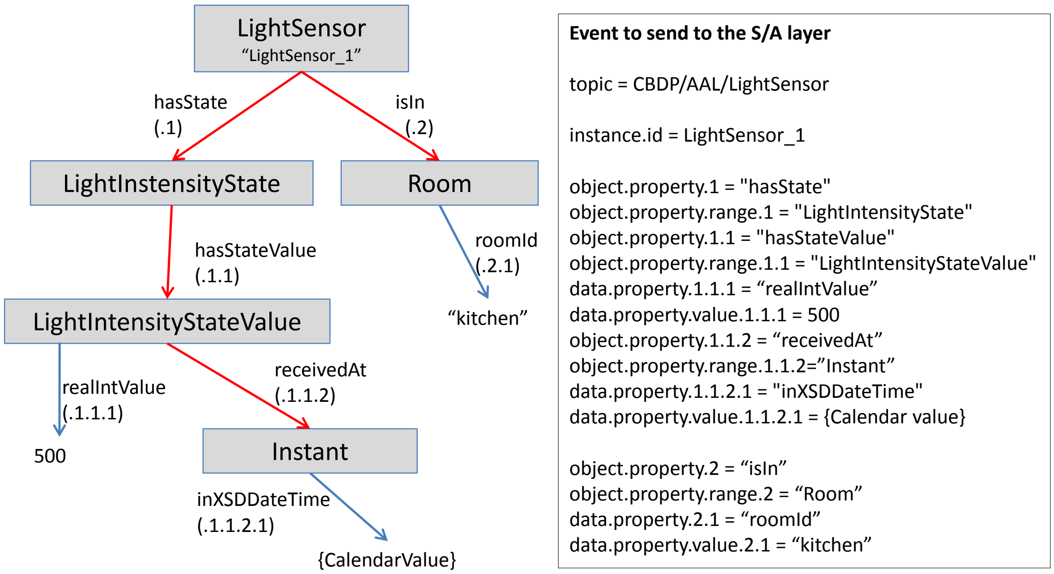

Figure 4. Example of an event containing sensor data.

for embedded systems that can be deployed on resource-constrained targets while providing real-time and safety guaranties. This demonstrates the adequacy of CBDP for its target applications, user assistance in ambient environments, i.e., in non computer-centric settings.

IV. Diagnosis

Ultimately, the goal of any AAL application is to activate some actuators, based on data provided by some sensors. However, sensors and actuators may suffer failures. Therefore the system should check autonomously whether the intended actions are performed correctly.

A. Rationale

In software, mechanisms such as exceptions and error codes report whether a procedure executes successfully or not. Likewise, an actuator can provide a return code, but generally this reflects only the way the orders are transmitted to the actuator, not their actual execution. For instance, when the system activates a light bulb, it receives an acknowledgement that confirms the switch-on of the electrical circuit, but this does not necessarily mean that the bulb is really on (the bulb may be damaged for instance). To address the issue, control theory could allow one to pre-determine closed control loops using designated sensors. However, the particularity of ambient systems is that physical resources, mainly sensors and actuators, are not necessarily known at design time, but are dynamically discovered at run-time, so such control loops cannot be pre-determined.

Therefore, a reliable AAL application needs a way to assess at run-time the status of its sensors and actuators. We propose an approach in which the system relies only sensors already available, thereby not requiring the addition of specific devices for diagnosis purposes. The sensors that may be used to perform diagnosis are discovered at run-time. When a sensor measures a physical parameter, the system may deduce sensor/actuator "health" status by comparing actual values with *expected* sensor values.

To achieve this, we propose a diagnosis framework in which the characteristics of actuators and sensors, as well as the *physical effects* involved, are precisely described. The following paragraphs provide a short summary of our approach; refer to [12] for more details.

B. Modeling physical effects

Effects are modeled in order to simulate the physical consequences of actions in an ambient environment. Each effect is characterized by a set of properties: some *define* the effect (at the source actuator, e.g., the light intensity *emitted by a light bulb*), some are *observable* by a sensor (e.g., the light intensity *received by a light sensor*).

Depending on the application's needs, an effect can be defined at various levels of granularity. For instance, the light emitted by a light bulb could be modeled either using classical laws of physics for light propagation, or using a simple boolean law ("if a light bulb is on in a room then the light sensors that are in that room should detect light").

C. Using effects for linking actuators to sensors

As ambient systems are highly dynamic, one cannot explicitly link related sensors and actuators. The concept of effect allows for easy decoupling of devices, as illustrated by Figure 5. An actuator class is linked to the effects it may potentially produce. Similarly a sensor class is linked to at least an effect property. At a generic level, there is a link between a given effect (e.g., emission of light) and

 ISBN: 978-1-61208-235-6

the corresponding detectable properties (e.g., light intensity) through the *hasProperty* relation.

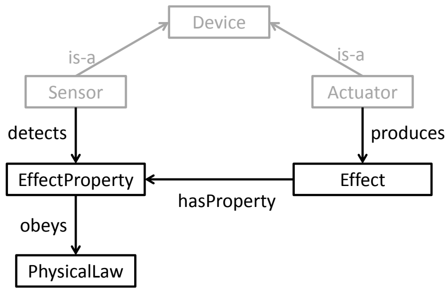

Figure 5. The diagnosis framework adds to the general-purpose ontology (gray part at the top) a few concepts to describe the effects and their detectable properties.

Knowing the effects produced by any actuator in the system, and knowing the effect properties sensed by any sensor in the system, it is therefore possible to determine and update at run-time the links between actual sensors and actuators. Hence it is possible to compute the expected readings of the sensors using physical laws. Once the expected results have been determined, the system checks if they are consistent with the actual readings.

V. EXPERIMENTATION

This section introduces a complete AAL scenario in which the CBDP framework is able to automate tasks, it shows how diagnosis is performed, and it describes the experiments.

A. *Use case: automatic light switch*

We propose the following experimental scenario. It takes place in a bedroom with a controlled lamp, a light sensor and a presence sensor: "if the ambient light level is under a threshold (specified in the Digital Personality of the user) and if the user is present in the room, then the light must be turned on". Although simple, this scenario demonstrates all the aspects of the system: sensor data gathering, reasoning, command of actuators and diagnosis.

Let us suppose that the light level in the room is 80 lux and then the user comes in. His Digital Personality states that he wants the lamp to be on when the light level is under 100 lux. The system takes the following steps:

1) The current light level (80 lux) has already been detected and updated in the ontology. When the user enters the bedroom, the presence sensor sends a notification to the driver through the Zigbee network. The driver sends then an event to the framework and the ontology is updated accordingly.
2) The framework detects that the value of a PresenceSensor has changed in the ontology, so the following rule must be evaluated (cf. III-B):

```
IF a LightSensor value is <
   {userPreference in the Digital Personality}
   AND a PresenceSensor detects somebody
   AND the LightSensor, the PresenceSensor
    and the LightActuator are in the same room
THEN Turn the LightActuator on
```

This rule is written here in pseudo-natural language for the sake of simplicity; in practice it is expressed in the formal syntax specific to the Jena reasoning engine as shown on Figure 6.
The reasoning engine reads the current light level, the current presence status and the user preferences in the ontology. The premises of the rule are true, so the conclusion must be executed. To determine which rules to apply, Jena uses a classical *forward chaining* reasoning algorithm.
3) Therefore a new statement is added in the ontology: {LightActuator, hasCommand, "on"} (cf. III-C and III-D). In consequence, the framework sends an event to the driver to indicate that the LightActuator must be turned on.
4) The driver commands the light actuator through the Zigbee network. This actually turns the light on.

```
[CMD_LIGHT_ON:
    (?MS RDF:type AMI:PresenceSensor),
    (?LS RDF:type AMI:LightSensor),
    (?LA RDF:type AMI:LightActuator),
    (?R RDF:type ?RT),
    (?RT RDFS:subClassOf AMI:Room),
    (?MS AMI:isIn ?R),
    (?LS AMI:isIn ?R),
    (?LA AMI:isIn ?R),
    (?DP RDF:type AMI:AAL_DP),
    (?DP AMI:isCurentDP ?curDP),
    equal(?curDP,'true'),
    (?DP AMI:low_AAL_LightThreshold ?LLT),
    (?MS CORE:realStateStringValue 'personInside'),
    (?LS AMI:realIntValue ?LMV),
    lessThan(?LMV,?LLT),
    (?F RDF:type AMI:OnOffFunctionality),
    (?LA AMI:hasFunctionality ?F),
    (?C RDF:type AMI:OnCommand)
                        -> (?F AMI:hasCommand ?C) ]
```

Figure 6. Example of rule ("turn the light on") expressed in Jena's syntax.

B. *Diagnosis*

At this point, the framework performs diagnosis so as to determine if the action has been executed correctly. The LightActuator is a "light effect" producer; the LightSensor measures the "light intensity" value of "light effect". They are in the same room, so a link between them is deduced automatically. Moreover, if some position reporting system is available, then the physical law associated with "light effect" (that calculates the light intensity) takes into account the actuator-sensor distance. The steps go on like this:

5) The diagnosis framework calculates the *expected light*

level at the light sensor by applying the formula associated with "light effect". The result is 120 lux.

6) The light level actually measured by the light sensor is still 80 lux, so the system deduces that there is a failure. The source of the failure (sensor or actuator) is *a priori* known with a limited probability only, but a second sensor in the room may increase it.
7) The system finds it most probable that the bulb is burnt out. An error notification is generated so that the user 1) confirms the cause the problem, and 2) possibly to fixes it (often, even an elderly person is capable of replacing a light bulb). For a discussion on the acceptability of notifications in a home environment, see for instance [13].

C. Implementation and Results

This experiment uses the standard CBDP framework, with a bundle containing its specific rules. The experiment was conducted in two ways:

- using a simulator of the sensors, actuators and physical environment,
- using physical devices in an actual room.

Figure 7 shows the interface of the simulation environment. The experimenters can act on the light level of the sun, on the motion sensor, and they can also introduce a defect in the light bulb. Both in simulation and in real conditions the system displays a message with the current diagnosis (Figure 8). The tests performed showed that the example runs as expected.

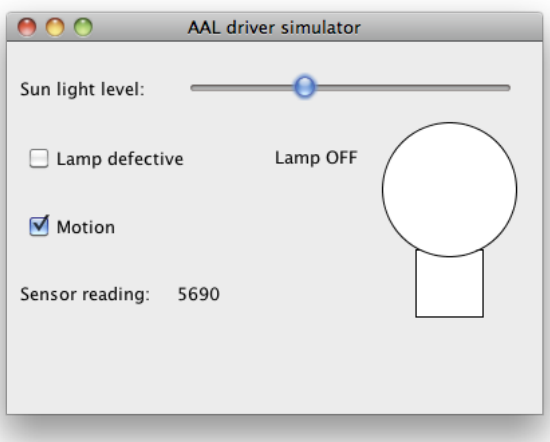

Figure 7. Interface of the simulated environment.

VI. Related Work

Ontologies are often at the heart of ambient-intelligent systems, and especially AAL systems, such as in OASIS [4]. In 2003, CoBrA (Context Broker Architecture) was an ontology-based framework for ambient settings [14].

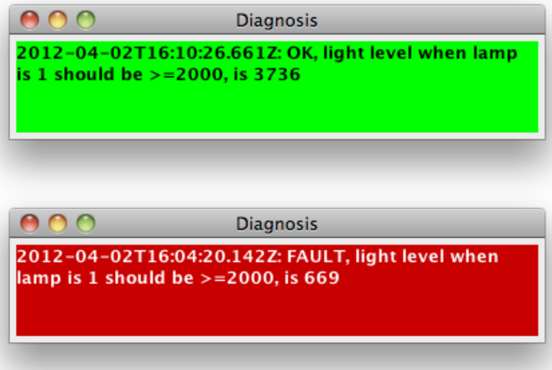

Figure 8. Window showing the results of the diagnosis.

In 2004, SOUPA (Standard Ontology for Ubiquitous and Pervasive Applications) [15] was one of the first attempts to define an application-agnostic ontology for ambient systems, but it is specifically aimed at agent-based architectures. More recently, Paganelli et al. [16] introduces a tele-health platform, which is based on an ontology for describing context and medical conditions. The SOPRANO project [17], [18] defines a specific ontology that serves as a unifying vocabulary between software components. In our work, the ontology is specifically used to personalize the system: it stores preferences, and contains application-specific modules. Moreover, we not only reason to infer new facts about context as done in many platforms [19], but also to trigger application-specific behavior, and to actually *trigger actions*, i.e., send commands to actuators. This makes the framework flexible and allows the easy integration of additional services such as the diagnosis framework described in Section IV.

Our choice of using the OSGi middleware was motivated by previous successful attempts in the field of ambient intelligence, such as in the AMIGO IST project [20]. CBDP's generalized reliance on ontologies makes the use of OSGi very consistent with the rest of the framework.

Some works focus on ontologies for specific domains. For instance, Hois [21] describes a well-grounded framework for the description of spatial relationships and spatial reasoning. This kind of contributions could be integrated into the CBDP framework, due to the reusable nature of ontologies.

VII. Conclusion and Future Work

We have presented a complete framework that supports the creation of AAL applications. This framework is based on the use of an ontology at the core of the system. This ontology contains application-specific knowledge and stores user preferences ("Digital Personality"). Besides it handles all the run-time information flows: it aggregates sensor data, allows rules to be applied on this data so as to generate commands, stores the commands, and provides the commands to the actuators.

Using an ontology allows one to specify the behavior of an AAL application in terms of easy-to-write logic rules. These rules can rely on any piece of knowledge present in the ontology, therefore they are not limited in any way by the core ontology that comes with the CBDP framework. Such extensibility is made easy by the use of widespread knowledge engineering standards, namely RDF/OWL.

The other significant contribution of this paper is the diagnosis framework that monitors the run-time behavior of an AAL system by observing changes in the ontology. Currently we take into account only the current state of the system. In reality, the relevant measure might not be the current absolute value of a physical parameter, but rather its *relative evolution*. For instance, when light is switched on, it may be most relevant to consider the *relative increase of the light level*, as the absolute value may vary other time without any action being taken (depending of the intensity of the sun for instance). This prompts us to introduce *dynamics* in the diagnosis framework. Likewise, some physical laws may depend upon quantitative *time* (for instance, the effect of a radiator in an initially chilly room is a slow increase of temperature over time). This is currently being investigated.

We also plan to test such a system in real scale, for example at the homes of elderly people. This will allow us to refine the rules that define the system behavior.

Acknowledgement

This work has been performed within the CBDP project, a project co-funded by the European Union. Europe is involved in Région Île-de-France with the European Regional Development Fund.

References

[1] G. Van den Broek, F. Cavallo, L. Odetti, and C. Wehrmann, “Ambient Assisted Living Roadmap,” AALIANCE, Tech. Rep., 2009.

[2] CBDP project description. Accessed on 18 July 2012. [Online]. Available: http://projects.celtic-initiative.org/cbdp/

[3] N. Noy and D. McGuinness, “Ontology development 101: A guide to creating your first ontology,” Stanford University, Tech. Rep., 2001.

[4] J. Bateman, A. Castro, I. Normann, O. Pera, L. Garcia, and J.-M. Villaveces, “OASIS common hyper-ontological framework (COF),” OASIS Project, Tech. Rep., 2009.

[5] G. Antoniou and F. van Harmelen, “Web ontology language: OWL,” in *Handbook on ontologies*, S. Staab and R. Studer, Eds. Springer, 2009, pp. 91–110.

[6] D. Bonino and F. Corno, “DogOnt–ontology modeling for intelligent domotic environments,” *The Semantic Web-ISWC 2008*, pp. 790–803, 2008.

[7] Jerry R. Hobbs and Feng Pan. (2006) Time Ontology in OWL. Accessed on 18 July 2012. [Online]. Available: http://www.w3.org/TR/owl-time/

[8] OSGi Alliance, *OSGi service platform, release 3*. IOS Press, Inc., 2003.

[9] J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson, “Jena: implementing the semantic web recommendations,” in *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*. ACM, 2004, pp. 74–83.

[10] A. Chandra and D. Harel, “Horn clause queries and generalizations,” *The Journal of Logic Programming*, vol. 2, no. 1, pp. 1–15, 1985.

[11] Atego. Aonix PERC product description. Accessed on 18 July 2012. [Online]. Available: http://www.atego.com/products/aonix-perc/

[12] A. Mohamed, C. Jacquet, and Y. Bellik, “Real-time Diagnosis of Ambient Environments Using a Modeling of Physical Effects Combined with Temporal Logic,” in *Proceedings of UBICOMM 2011*. IARIA, Nov. 2011, pp. 145–151.

[13] M. Vastenburg, D. Keyson, and H. Ridder, “Considerate home notification systems: a field study of acceptability of notifications in the home,” *Personal and Ubiquitous Computing*, vol. 12, no. 8, pp. 555–566, 2008.

[14] H. Chen, T. Finin, and A. Joshi, “An intelligent broker for context-aware systems,” in *Adjunct Proceedings of Ubicomp*, vol. 3, 2003, pp. 183–184.

[15] H. Chen, F. Perich, T. Finin, and A. Joshi, “SOUPA: standard ontology for ubiquitous and pervasive applications,” in *Mobile and Ubiquitous Systems: Networking and Services (MOBIQUITOUS), 2004*. IEEE, 2004, pp. 258–267.

[16] F. Paganelli and D. Giuli, “An Ontology-Based Context Model for Home Health Monitoring and Alerting in Chronic Patient Care Networks,” in *Proceedings AINA 2007*. IEEE Computer Society, 2007, pp. 838–845.

[17] M. Klein, A. Schmidt, and R. Lauer, “Ontology-centred design of an ambient middleware for assisted living: The case of SOPRANO,” in *30th Annual German Conference on Artificial Intelligence (KI 2007), Osnabrück*, 2007.

[18] P. Wolf, A. Schmidt, and M. Klein, “SOPRANO-An extensible, open AAL platform for elderly people based on semantical contracts,” in *AITAmI, ECAI’08*, 2008.

[19] A. Bikakis, T. Patkos, G. Antoniou, and D. Plexousakis, “A survey of semantics-based approaches for context reasoning in ambient intelligence,” *Constructing Ambient Intelligence*, pp. 14–23, 2008.

[20] F. Ramparany, R. Poortinga, M. Stikic, J. Schmalenstroer, and T. Prante, “An open context information management infrastructure the ist-amigo project,” in *Proceedings of IE 2007, the 3rd IET International Conference on Intelligent Environments*. The IET, 2007, pp. 398–403.

[21] J. Hois, “Modularizing spatial ontologies for assisted living systems,” in *Proceedings of the 4th international conference on Knowledge science, engineering and management*, ser. KSEM’10. Springer-Verlag, 2010, pp. 424–435.