Noname manuscript No.
(will be inserted by the editor)

Design and Evaluation of an Ambient Assisted Living System Based on an Argumentative Multi-Agent System

Andrés Muñoz · Juan Carlos Augusto · Ana Villa · Juan Antonio Botía

the date of receipt and acceptance should be inserted later

Abstract This paper focuses on Ambient Assisted Living systems employed to monitor the ongoing situations of elderly people living independently. Such situations are represented here as contexts inferred by multiple software agents out of the data gathered from sensors within a home. Sensors can give an incomplete, sometimes ambiguous, picture of the world, hence they often lead to inconsistent contexts and unreliability on the system as a whole. We report on a solution to this problem based on a multi-agent system where each agent is able to support its understanding of the context through arguments. These arguments can then be compared against each other to determine which agent provides the most reliable interpretation of the reality under observation.

Keywords Ambient Intelligence · Assisted Living · Multi-Agent System · Argumentation

1 Introduction

Ambient Assisted Living (AAL) is an emerging area within Ambient Intelligence [7] mainly aimed at enabling elderly people to live more independently and for longer at their homes. The monitoring of their activities through AAL systems allows the detection of undesired situations they might suffer. The notifications of these situations can be used to trigger an emergency mechanism which eventually may help in saving lives.

The design of AAL systems is normally based on the use of infrastructures provided by Intelligent Environments, see [2] for an example. Such infrastructures are composed of physical devices for sensing and actuation coordinated by a software layer which offers both an intelligent analysis of the information gathered through the devices and an intelligent-decision making to drive the actuation of the AAL system.

A recurrent problem these systems have to deal with is the detection of complex, sometimes inconsistent, situations from simple sensors events. For example, if the person does not react to a door bell ring, it may indicate s/he has a serious problem but it could also be simply because she is having a bath. This paper proposes an AAL system based on a multi-agent architecture responsible for analyzing the data produced by different types of sensors and inferring what contexts can be associated to the monitored person. To this end, Semantic Web ontologies [3] are adopted to model sensors events and the person's context. The agents use rules defined on such ontologies to infer information about the current context. In case that agents discover inconsistent contexts, we use of argumentation techniques [4] to disambiguate the situation by comparing the arguments each agent creates to support their points of view. As a result, the AAL system provides an effective monitoring platform even in ambiguous situations.

The reminder of this paper firstly reviews related work in the AAL area. The following section shows how sensor events and contexts are represented by means of ontologies. Next, the argumentative multi-agent architecture for the proposed AAL system is explained. Finally, we show the results of the system at work.

A. Muñoz · J. A. Botía
Dpto. de Ingeniería de la Información y las Comunicaciones
University of Murcia, C.P. 30100, Murcia, Spain
E-mail: amunoz@um.es|juanbot@um.es

J. C. Augusto
School of Computing and Mathematics
Computer Science Research Institute
University of Ulster, BT37 0QB Newtownabbey, United Kingdom
E-mail: jc.augusto@ulster.ac.uk

A. Villa
Ambiental Intelligence & Interaction, SLL (Ami2)
Ed. CEEIM Modulo 11, Campus Espinardo, 30100 Murcia, Spain
E-mail: ana.villa@ami2.net

2 Related Work

Intelligent Environments depend on artificial sensing. Current state of the art in artificial sensing is still *incomplete*, giving only a partial picture of an environment; *unreliable*, due to technical faults or misplacing of sensors may mean they do not deliver data continuously; and *corrupted*, interference of other devices may cause the deliverance of wrong data. The frequency and harmful impact of these problems on the system as a whole are relevant enough to be a matter of concern within any intelligent environment.

Computer Scientists have encountered similar problems in other real-time application domains, leading to the development of methods to deal with uncertainty, ambiguity and contradiction. Researchers in the area of Intelligent Environments have applied those methods to the problem of reasoning with "noisy" data based on fuzzy techniques, probabilistic systems or Markov-based models [1]. These approaches are eminently numerical and hence require the explicit estimation and input of probabilities and specific numbers associated to the diversity of possible situations expected in the modeled system [11].

Our method based on argumentation [4] provides a complementary approach to deal with uncertainty, ambiguity and contradiction in Intelligent Environments. This approach is more qualitative in nature than the aforementioned ones, and therefore easier to be assessed by people who are less specialized on the intrinsic algorithms utilized in them. Argumentation is modeled on the dialectical process followed by humans when solving conflicting situations. Selection amongst alternative viewpoints about a situation is based on well-known logical abstract reasoning patterns as well as on domain specific criteria (e.g., trust on the source of information, objective statistics linked to a specific context, etc.).

3 Representing Events and Contexts in the AAL system

AAL systems rely on smart homes equipped with different types of sensors to obtain information about the person being monitored (*occupant* henceforth). Depending on the events to be collected, and also on privacy and unobtrusiveness issues, the sensors employed vary from simple movement detectors, pressure pads in sofas and beds, on/off switch sensors attached to electric appliances, taps, etc., to video cameras (VC) that record symbolic information instead of real images or accelerometers worn by the occupant to indicate her position (i.e., standing up, sitting down or lying down). The collected events are then utilized by the AAL system to determine the occupant's context.

In our AAL system, smart home structures, events and contexts are represented by means of *ontologies*. They are formal descriptions of domains through a set of concepts and relationships between them. Specific situations in these domains are modeled by means of *assertions* or statements about instances of such concepts. OWL (Web Ontology Language) [8] is the standard language adopted to describe ontologies in computational systems. In particular, the ontologies developed here for representing smart homes, events and contexts are written in OWL DL, an expressive and decidable version of this language (DL stands for Description Logics). The rest of this section highlights the main characteristics of the ontologies developed for our AAL system.

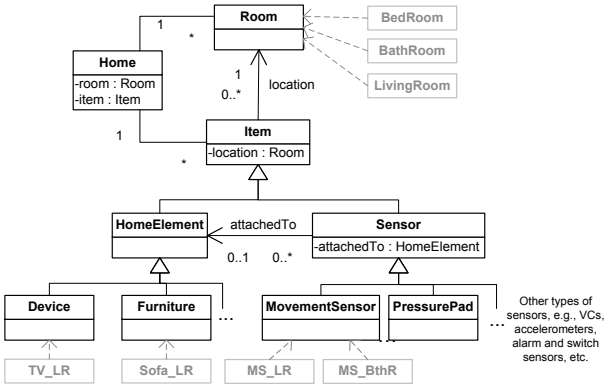

Fig. 1 A smart home ontology, *SmartHome_Ont*, partially represented through an UML model.

Figure 1 depicts a simplified view of our ontology *SmartHome_Ont* as a UML model. It is composed of concepts that represent rooms, home elements and sensors along with relationships among them. Gray boxes represent instances of these concepts. This ontology allows the description of any smart home structure containing different elements and sensors. New types of sensors can be easily modeled by extending the concept `Sensor`.

Events and contexts are represented by means of their respective ontologies *Event_Ont* and *Context_Ont* depicted in Fig. 2. Note that the definitions of both `Event` and `Context` concepts are based on SmartHome_Ont concepts (relationships with such concepts are not shown here for simplicity). Let us explain these two concepts.

Firstly, the concept `Event` (see Fig. 2(a)) represents each event with the type of sensor producing it, the room and/or element where it is detected, its value and a time stamp. Note that there is a specific kind of event for each type of sensor given in SmartHome_Ont. The current version of Event_Ont contains concepts to represent events from movement sensors, pressure pads, switch sensors, accelerometers, VCs, light sensors and alarm sensors, the latter generating *unattended events* to indicate an action which has not been performed by the occupant within the expected time, e.g., answering a phone call or a door bell ring.

Each specific instance of `Event` is represented with a unique ID, e.g. ev_1, and it is associated to the corresponding type of event through a conceptual assertion, e.g. `Mov_Ev`(ev_1)

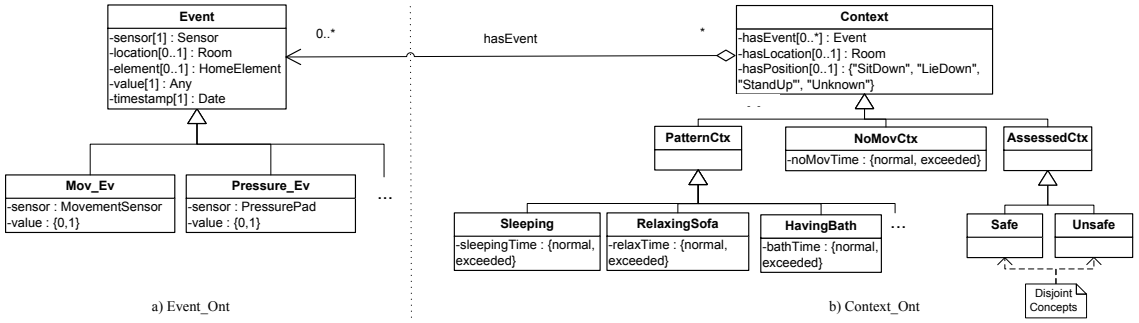

Fig. 2 Partial UML representations of *Event_Ont*(a) and *Context_Ont*(b) ontologies.

given that ev_1 has been obtained from a movement sensor. Its attributes are stated through binary relational assertions, e.g. `value`(ev_1,0) and `location`(ev_1,`Bedroom`) indicate that no movement has been detected in the bedroom.

Secondly, the concept `Context` represents the current occupant's situation (see Fig. 2(b)). The multi-valued attribute `hasEvent` associates each context with the set of events happening in it. Moreover, this concept also indicates the occupant's location and position. A context can be specialized in `PatternCtx`, `NoMovCtx` and `AssessedCtx` subtypes. Thus, a pattern context refers to those deduced from a combination of events –possibly in a specific order– and occupant's location and position that match with an activity pattern, e.g. sleeping, watching TV, etc. Note that these contexts have an attribute to indicate if the normal amount of time expected for that activity has been surpassed. On the other hand, `NoMovCtx` represents a situation where no movement is detected in the occupant's location, also with an attribute to indicate the amount of time in this situation. Besides, a context can be assessed as *safe* or *unsafe* to indicate whether the occupant could be in a problematic situation or not. Section 4 shows how context information is implied by the multi-agent architecture of the AAL system.

Each instance of `Context` has a unique ID, e.g. Ctx_1, and its types are stated through conceptual assertions, e.g. `Sleeping`(Ctx_1). The set of events and the occupant's location and position associated to a context are stated through relational assertions, e.g. `hasEvent`(Ctx_1,ev_1).

In our system, a context can have at most one assessment type and one occupant's location and position. If the same context is classified as safe and unsafe, or more than one location or position has been detected in it, an inconsistency arises in the system. This inconsistency is automatically detected due to the formal features of the OWL DL language, which allows to define the concepts `Safe` and `Unsafe` as disjoints. Similarly, the attributes `hasLocation` and `hasPosition` are declared to be *functional*, i.e., they can have at most one value for the same context.

It is important to clarify that the context information is assumed to be obtained when the occupant is the only person detected at home. Therefore, if the occupant leaves the house or more than one person is known to be at home, the AAL system notifies it to the caregivers and switches to a standby phase until the occupant is detected to be the only person at home again. These cases are handled by sensors in the main door not explained here.

Using OWL DL ontologies to represent smart homes, events and contexts has multiple advantages: (a) it allows an expressive modeling of this information, (b) the ontologies are used as a mechanism to share knowledge within the multi-agent architecture of the AAL system in a standard manner, and (c) their formal features enable the validation of the knowledge held in the system in order to detect conflicting contexts.

4 Argumentative Multi-Agent Architecture

This section explains the AAL system architecture to evaluate the occupant's context out of the collected sensor data. To this end, a distributed inference process is performed by agents taking into account the occurrence of conflicting contexts, solved by argumentation. Next we provide a brief introduction on the underlying argumentation theory and the design of the argumentative multi-agent architecture.

4.1 Background on Argumentation Systems

Argumentation systems differ from other knowledge systems on the capability of generating justifications –i.e., arguments– to support inferences which can, potentially, be inconsistent. The use of heuristics allows to take decisions on any inconsistency detected by establishing which argument is more believable. For technical and formal details about the definitions and the argumentation process described here, the reader is referred to [9].

In the context of our argumentation system, an argument is defined as the justification for an assertion through a deductive inference process. Thus, we define an argument as (ϕ, S), where ϕ is the conclusion, i.e. the assertion being justified, and S=$\{\Phi, r\}$ is the support set of ϕ, i.e., a set of assertions Φ and a rule r such as Φ is the minimal set that fulfills

the antecedent of r so as its consequent is instantiated to ϕ (we assume rules with only one consequent). Assertions and rules are expressed using the vocabulary on events and contexts modeled through the ontologies described in Section 3 (some examples of arguments will be given in Section 4.2). Henceforth we will use $A_1, A_2, \ldots,$ to denote arguments.

Due to the formal features in the OWL DL ontologies used to express arguments, the agents of the AAL system are able to automatically process such arguments in order to detect conflicts between them. Thus, an argument A_1=(ϕ, S_{A_1}) is in conflict with an argument A_2=(φ, S_{A_2}) by *rebutting* or *undercutting* it. In the first case, the union of their conclusions ϕ and φ produces an inconsistency in the ontology. In the second case, the union of ϕ and one assertion of S_{A_2} produces the inconsistency.

Apart from the notion of conflict between a pair of arguments, it is also necessary to define the *defeat* relationship among them, i.e., which argument is more plausible. Thus, A_1 can defeat (is more plausible than) A_2 in two ways: (1) when A_1 undercuts A_2, since this attack invalidates a premise needed by A_2 to support its conclusion; and (2) A_1 rebuts A_2 and besides A_1 is preferred to A_2 according to some criteria and A_2 does not undercut A_1. There exist several types of criteria to decide which argument is more plausible when they are rebutting each other, such as specificity of information contained in the premises, rule precedence or agent priorities [4]. Sections 4.2.2 and 4.2.3 show the criteria we use in the AAL system for rebutting arguments.

Finally, notice that the defeat relationship is defined for two arguments solely. Since a defeating argument can be defeated by another argument and so on, it is also needed a procedure to establish the eventual status of each argument on the basis of all interactions among them. To this end we build a tree that represents all the relationships among arguments. Its root contains the argument A_I whose status needs to be determined. Such a tree is evaluated by means of an algorithm which marks each argument in the tree as *in* or *out* according to its defeat relationships (see [9] for details). A_I is accepted if the root is labeled as *in*, and rejected if the root is labeled as *out*. It is also possible to find A_I as *undecided* when the status of the root cannot be determined, i.e., its evaluation depends on arguments which either are incomparable or have an equal strength.

4.2 Architecture Design

The AAL system architecture is organized into three layers: Event Management System (EMS), Context, and Assessment. Fig. 3 shows the overall architecture. The rest of this section describes each layer in detail.

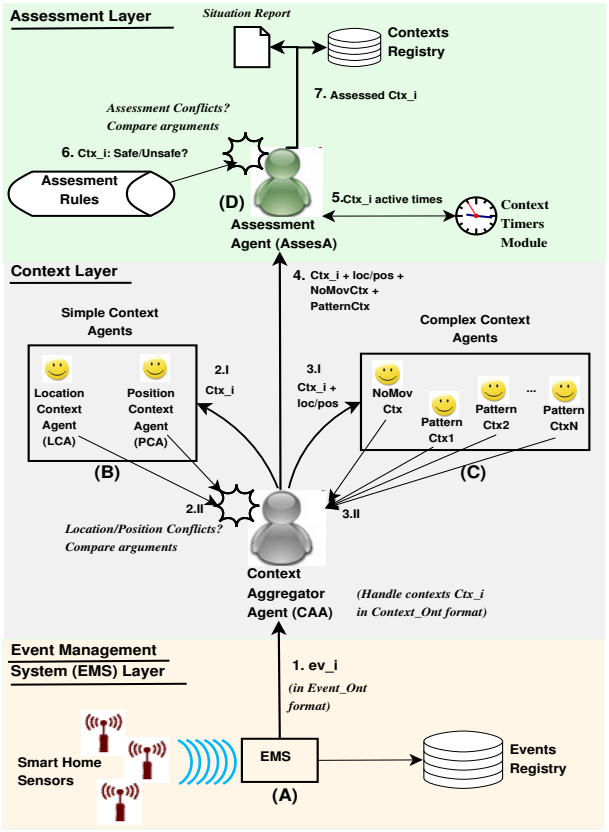

Fig. 3 Argumentative multi-agent architecture for the AAL system.

4.2.1 EMS Layer

This layer is in charge of receiving raw data from sensors and converting them into events represented according to the ontology Event_Ont. To this end, each type of sensor is associated to a software *adapter* that transforms its values into semantic annotations. The adapters are registered in the Event Management System (see (A) in Fig. 3), which generates the corresponding ontology assertions to represent events from the semantic annotations. In order to obtain the location and house object associated to an event, the EMS queries the SmartHome_Ont ontology to retrieve such information according to the sensor which generated the event.

Once a new event has been generated, the EMS stores it in the Event Registry to keep a log of events. Finally, the EMS notifies the Context Aggregator agent about the new event (see step 1 in Fig. 3), which determines the occupant's context as explained in the next section.

4.2.2 Context Layer

This layer obtains information about the occupant's context. As shown in Section 3, such information includes the occupant's location and position along with more complex situations such as to be sleeping, to be resting on the sofa, etc. Note that this layer only generates a picture of the current

context, while its evaluation as safe or unsafe is performed in the Assessment Layer to be explained in the next section.

Our approach to obtain the information related to a context is to use a different agent for managing each type of information. In particular, we will have an agent to infer location information, another one for position information, one agent for each pattern context specified in Context_Ont (see Fig. 2(b)), and a final one to discover the absence of movement in the occupant's location. These agents own the specific knowledge to infer the required information and build arguments supporting it. As a result, the task of obtaining context information is executed in a distributed and efficient manner, and it can be easily extended with more agents to infer new information. Let us now see the process performed in this layer to infer the occupant's context.

Firstly, the *Context Aggregator Agent* (CAA) uses Context_Ont to represent the current context as follows. It starts creating an instance Ctx_i of the concept `Context` when a new event ev_i is received from the EMS, where i is the sequence index of these elements. In this manner, CAA creates a new context to represent the situation given by the arrival of a new event. Then, Ctx_i is associated to its set of events Ev_i using the relationship `hasEvent`, where Ev_i contains ev_i and all the events of the previous context Ctx_{i-1}, contained in Ev_{i-1}, that are not updated by ev_i. Update of events appears when ev_i and one event $ev_{i'}$ in Ev_{i-1} are produced by the same sensor and they indicate different values. For example, if ev_i and $ev_{i'}$ are generated by the movement sensor in the bedroom with values 1 (movement detected) and 0 (no movement detected) respectively, then ev_i updates $ev_{i'}$.

Once the current context Ctx_i has been generated, the CAA sends it together with its associated set of events to the Simple Context Agents group (see (B) and step 2.I in Fig. 3). This group is composed of the Location Context Agent (LCA) and Position Context Agent (PCA), which are responsible for inferring the respective occupant's location and position according to their rules. Hence, LCA has the next two rules to determine the occupant's location based on movement sensors and pressure pads (it has other rules based on other types of sensors):

$$R_{LC1}: \texttt{Context}(?c) \wedge \texttt{hasEvent}(?c,?e) \wedge \texttt{Mov\_Ev}(?e) \wedge \texttt{value}(?e,1) \wedge \texttt{location}(?e,?l) \Rightarrow \texttt{hasLocation}(?c,?l)$$

$$R_{LC2}: \texttt{Context}(?c) \wedge \texttt{hasEvent}(?c,?e) \wedge \texttt{PressurePad\_Ev}(?e) \wedge \texttt{value}(?e,1) \wedge \texttt{location}(?e,?l) \Rightarrow \texttt{hasLocation}(?c,?l)$$

Agents' rules are defined with assertions expressed by means of concepts and relationships given in Context_Ont and Event_Ont. They are composed of an antecedent expressed as a conjunction ($\wedge$) of such assertions with variables, denoted with '?', and a consequent with one assertion. If all assertions in the antecedent are matched with the context information maintained by the corresponding agent, the assertion in the consequent is inferred instantiating its variables with the matching values used in the antecedent. These rules are evaluated by a rule engine included in each agent. Details about the specific implementation of this kind of rules and rule engine are given in Section 5.

Now, supposing LCA receives the context Ctx_i containing the event $\texttt{Mov\_Ev}(ev_j)$ from the movement sensor in the hall with value 1, it builds the following argument:

$$A_{LC1}: (\texttt{hasLocation}(Ctx_i, Hall), \{\texttt{Context}(Ctx_i), \texttt{hasEvent}(Ctx_i, ev_j), \texttt{Mov\_Ev}(ev_j), \texttt{value}(ev_j,1), \texttt{location}(ev_j, Hall), R_{LC1}\})$$

After receiving the LCA's arguments with the inferred location for Ctx_i (see step 2.II in Fig. 3), the Context Aggregator Agent has to handle one of these three situations: (a) no location information has been inferred, then Ctx_i is associated to the location in Ctx_{i-1}; (b) a unique location has been inferred, then Ctx_i is associated to it; (c) more than one location has been inferred, then CAA uses the received arguments for each location to decide which one should be associated to Ctx_i. Note that the case (a) happens when Ctx_i does not have enough events to discover a location, while the case (c) happens when it has two or more events generated by sensors that enable different locations being inferred, e.g., movement sensors, pressure pads or VCs. As an example of this latter case, suppose that Ctx_i also contains the event $\texttt{PressurePad\_Ev}(ev_{j'})$ from the sofa in the living room with value 1. Then, LCA generates the following argument A_{LC2} for that location through the rule R_{LC2}:

$$A_{LC2}: (\texttt{hasLocation}(Ctx_i, LivingRoom), \{\texttt{Context}(Ctx_i), \texttt{hasEvent}(Ctx_i, ev_{j'}), \texttt{PressurePad\_Ev}(ev_{j'}), \texttt{value}(ev_{j'},1), \texttt{location}(ev_{j'}, LivingRoom), R_{LC2}\})$$

Note that A_{LC1} and A_{LC2} rebut each other since their conclusions violate the functional restriction of `hasLocation` for the context Ctx_i according to the Context_Ont ontology. CAA detects this kind of conflicts using an OWL DL reasoner to validate the location assertions received from LCA. Location inconsistencies may be caused for many reasons like temporary malfunctions or bad positioning of sensors. For example, a movement sensor in a room A may be triggered by the occupant whilst still at another room B but near the door connecting both rooms. To solve these inconsistencies, CAA compares the arguments received for each inferred location using the argumentation process explained in Section 4.1 and considering the type of event in the premises of each argument as the criterion to decide which argument is the most reliable. For example, a criterion can state that locations inferred by events produced by a VC are more reliable than the ones based on pressure pads events, and in turn, these are more reliable than those inferred by movement sensor events. As a result, argument A_{LC2} is more reliable than argument A_{LC1}, and the location for Ctx_i is believed to be the living room. If all the arguments supporting conflicting locations are equally reliable, then Ctx_i is associated to the location in Ctx_{i-1}. The use of additional information such as the time stamp of events and the frequency of conflicts

related to the same type of event –denoting a malfunction of a sensor– are being currently taken into account in argument evaluation. The entire process described for inferring location information is similarly performed by PCA to generate position information using VCs and accelerometers, considering VC triggered events more reliable than accelerometer triggered events in case of conflict.

The next step is to determine some more complex information about the current context, the occupant's activity. To this end, CAA sends Ctx_i along with its inferred location/position information to the Context Complex Agents group (see (C) and step 3.I in Fig. 3). Each of the agents in this group is responsible for inferring a `PatternContext` type of Ctx_i as the ones shown in Fig. 2(b). Depending on whether the pattern can be recognized as a combination of events happening at the same time or as an ordered and temporal sequence of events, the agent's knowledge for it is expressed as a rule or as a finite state machine (FSM), respectively. As an example of the first case, the pattern `RelaxingSofa`, which represents the occupant reading, having a nap, etc. in the sofa placed in the living room, is inferred by an agent owning the following rule (similar rules including position information, TV on/off events, etc. may be also defined to obtain a more accurate context):

$$R_{RxSofa}: \texttt{Context}(?c) \wedge \texttt{haslocation}(?c, \mathit{LivingRoom}) \wedge \texttt{hasEvent}(?c, ?e) \wedge \texttt{PressurePad\_Ev}(?e) \wedge \texttt{value}(?e, 1) \wedge \texttt{element}(?e, \mathit{SofaLivingR}) \Rightarrow \texttt{RelaxingSofa}(?c)$$

PatternCtx agents based on rules generate arguments for the entailed activity in the same manner LCA and PCA do. Thus, the previous agent will create an argument in the form A_{RxSofa}:(`RelaxingSofa`(Ctx_i),{...}) when such a context Ctx_i satisfies the rule above. At the moment we have defined pattern contexts which are mutually exclusive, including those entailed through FSMs as explained next. Therefore, it is assumed that only one activity is associated to the current context and no conflicting activities can be detected.

Regarding the FSMs used in PatternCtx agents, they represent the sequence of situations –states in an FSM– to recognize an activity. In this paper we introduce them in an informal manner. Each FSM is composed of an *initial* state that indicates the starting situation of the activity, followed by zero or more *intermediate* states to represent the situations that lead to its recognition, next a *triggering* state that indicates when such an activity has been recognized, and finally an *end* state to indicate when it exits from being active. The transition from a state S_1 to the next state S_2 is performed when S_1 is *fulfilled*. It occurs when the current Ctx_i contains occupant's location/position information and a set of events that match the situation described in such a state. While an FSM is in the triggering state, its agent notifies the recognized activity pattern to the Context Aggregator Agent. When the end state is reached, the agent stops notifying this activity and the FSM is reset. Two descriptions of FSMs developed in the system are given next.

The first FSM recognizes the context `HavingBath` (here we use sensors which indicate the states of the bath tap and bath drain as open or closed). It is composed of an initial state representing the occupant located in the bathroom, followed by an intermediate state where the bath tap is opened and the bath drain is closed, after a few minutes the triggering state is reached when the bath tap is closed, and finally the end state is detected when the bath drain is opened. Observe that the triggering state –i.e., to close the bath tap– may have a time-limit condition for fulfilling it. If it is surpassed, the agent in charge of this FSM indicates this situation generating an unattended event for the bath tap (e.g., the bath is open for more than a limit of six minutes, due to the occupant has forgotten she was going to have a bath). The detection of the context `Sleeping` is also based on an FSM, which sequence of states starts with the occupant located in the bedroom, followed by the activation of the bed pressure pad, and reaching the triggering state when the bedroom lights are switched off. It changes to the end state when the lights are switched on or the bed pressure pad is released.

Apart from PatternCtx agents, there is also an agent in the Complex Context Agents group for inferring the context `NoMovCtx`, which represents the absence of movement in the room where the occupant is located in (see Section 3). To this end, this agent has the following rule:

$$R_{NM}: \texttt{Context}(?c) \wedge \texttt{hasLocation}(?c, ?l) \wedge \texttt{hasEvent}(?c, ?e) \wedge \texttt{Mov\_Ev}(?e) \wedge \texttt{value}(?e, 0) \wedge \texttt{location}(?e, ?l) \Rightarrow \texttt{NoMovCtx}(?c)$$

Assuming that the location of the context Ctx_i is stated in the living room and it has a movement sensor event ev_k with value 0, this agent generates the next argument for Ctx_i:

$$A_{NM}: (\texttt{NoMovCtx}(Ctx_i), \{\texttt{Context}(Ctx_i), \texttt{hasLocation}(Ctx_i, \mathit{LivingRoom}), \texttt{hasEvent}(Ctx_i, ev_k), \texttt{Mov\_Ev}(ev_k), \texttt{value}(ev_k, 0), \texttt{location}(ev_k, \mathit{LivingRoom}), R_{NM}\})$$

Once CAA has received all the information inferred by the Complex Context Agents group, it sends the current context Ctx_i with location/position information and the detected pattern activity and/or lack of movement to the Assessment Agent (see step 4 in Fig. 3), which establishes the safety of such a context as explained in the next section.

4.2.3 Assessment Layer

This layer evaluates the contexts received from CAA to decide whether the occurrence of unattended events –e.g., a door bell ring not answered– or an unusually prolonged lack of activity indicate the occupant may be experiencing a problem –e.g., fainted or feeling unwell– (*unsafe* contexts) or are due to other reasons –e.g., having a bath or sleeping– (*safe* contexts). Assessment of contexts is performed by the Assessment Agent (AssesA, see (D) in Fig. 3), which uses a timer module and a set of assessment rules to this end, apart

from the context information provided by the Context Aggregator Agent.

Firstly, AssesA calls the timer module to monitor the amount of active time of the PatternCtx and NoMovCtx situations associated to the received context (see step 5 in Fig. 3). The aim of this module is to control an excess of the expected times for such situations to be active, which may indicate an unsafe context (e.g., a fall could cause an excessive time for NoMovCtx). There are several approaches to calculate these times, depending on the daytime, occupant's location, etc. The interested reader is referred to [5] for details on these approaches.

While the monitored situations hold and their timings are normal, AssesA adds the assertions $\texttt{sleepingTime}(Ctx_i, normal)$, $\texttt{noMovTime}(Ctx_i, normal)$, etc. for each received Ctx_i (see Section 3 for details about these relationships). If a new context arrives in which a monitored situation does not hold any more, its timer is discarded. On the other hand, when a timer reaches zero, the timer module notifies AssesA and the agent updates the corresponding time attribute with the value *exceeded*. For example, assuming that the timer for sleeping reaches zero in the context Ctx_i, then AssesA states $\texttt{sleepingTime}(Ctx_i, exceeded)$.

When a new context arrives or it is changed by a timer notification, AssesA evaluates it by means of assessment rules (see step 6 in Fig. 3). These rules are divided into two groups according to the descriptions given at the beginning of this section: Unsafe-situation (US) rules to entail a context as unsafe, and Safe-situation (SS) rules to entail it as safe. Thus, the following rules belong to the US rule set:

$$R_{US1} : \texttt{NoMovCtx}(?c) \wedge \texttt{noMovTime}(?c, \text{“}exceeded''\text{)} \Rightarrow \texttt{Unsafe}(?c)$$

$$R_{US2} : \texttt{Context}(?c) \wedge \texttt{hasEvent}(?c, ?e) \wedge \texttt{Unatt\_Ev}(?e) \Rightarrow \texttt{Unsafe}(?c),$$

indicating that a context with an excessive lack of movement (R_{US1}) or unattended events (R_{US2}) is unsafe.

Whilst US rules express general situations deemed as unsafe, SS rules are defined as exceptions to them by adding new information that turns the context evaluation as safe. The following rules belong to the SS rules set:

$$R_{SS1} : \texttt{NoMovCtx}(?c) \wedge \texttt{noMovTime}(?c, \text{“}exceeded''\text{)} \wedge \texttt{Sleeping}(?c) \wedge \texttt{sleepingTime}(?c, \text{“}normal''\text{)} \Rightarrow \texttt{Safe}(?c)$$

$$R_{SS2} : \texttt{Context}(?c) \wedge \texttt{hasEvent}(?c, ?e) \wedge \texttt{Unatt\_Ev}(?e) \wedge \texttt{element}(?e, doorBell) \wedge \texttt{HavingBath}(?c) \Rightarrow \texttt{Safe}(?c)$$

They can be considered as exceptions to the previous US rules. Rule R_{SS1} indicates that an excessive lack of movement is safe if the occupant is in a normal-time sleeping context, where R_{SS2} entails an unattended door bell ring as safe if the occupant is having a bath.

AssesA builds different arguments through assessment rules to establish the safety of each context. Since arguments based on SS and US rules rebut each other, conflicting situations arise when both types of arguments are generated. For instance, an unattended door bell ring in a context Ctx_i generates the argument A_{US2}=$(\texttt{Unsafe}(Ctx_i), \{\ldots, R_{US2}\})$. However, if it is also known that the occupant is having a bath, the argument A_{SS2}=$(\texttt{Safe}(Ctx_i), \{\ldots, R_{SS2}\})$ is created to indicate that it is a normal behavior not to open the door in that situation.

AssesA detects these conflicts of assessments –using an OWL DL reasoner– since the concepts Safe and Unsafe are mutually exclusive in the Context_Ont ontology. Such conflicts are solved by using precedences among the rules employed in the arguments. Thus, an SS rule is more prevalent than a US rule if it is declared as an exception to the latter, since the SS rule contains more information. This criterion is related to a well-known comparison mechanism in argumentation called *specificity* [4]. Following the example above, A_{SS2} is more prevalent than A_{US2} since R_{SS2} is an exception to R_{US2}. As a result, Ctx_i is eventually evaluated as safe. Contrarily, if an argument A_{US} claiming an unsafe context is rebutted by an argument A_{SS} whose rule is not an exception to the rule in A_{US}, then A_{US} is more prevalent than A_{SS} and the context is stated as unsafe. In case that there is not enough information to classify a context as safe nor unsafe, it is considered that no problems have been detected.

Finally, AssesA registers each evaluated context in the Context Registry and generates a report indicating its safety type (see step 7 in Fig. 3). The arguments for each context are also included in the report in a suitable form to give a human-readable explanation of why the context has been deduced as safe or unsafe. In case of an unsafe context, AssesA notifies the caregivers through the corresponding mechanisms, e.g., via SMS or a phone call.

5 Evaluation

The AAL system proposed here has been deployed and evaluated within the Alerting Intelligent Devices project (DIA for its Spanish acronym) developed at the Ami2 company. The main goal of DIA is to use smart environments to detect alterations in the occupant's behavior and thus alerting to possible problems. A pilot study was performed in the first stage of this project aimed to detect periods of inactivity at home which may indicate some problems, e.g., the occupant has suffered a fall or stroke. However, these periods could occur because she is relaxing in the sofa or sleeping, which are situations to be detected as safe in this first stage.

The tests executed in the study consisted in monitoring three occupants living for one month in homes equipped with movement sensors in all rooms, pressure pads in beds and sofas, a light sensor in the bedroom, and a sensor in the main door to indicate if the occupant leaves/enters the house. A simple automaton and a set of timers were used in the pilot study to detect excessive inactivity (see [5] for details). Here we use the sensor data collected in this study to

reproduce them in our proposed AAL system with the aim of evaluating it. The next sections discuss the deployment of the system and the obtained results.

5.1 Deployment of the AAL system

The EMS of this system, implemented through OCP (Open Context Platform) [10], is an ontology-based context middleware which is able to handle sensor data and transform them into ontology assertions by using sensor adapters and the SmartHome_Ont ontology. OCP is based on the Jena framework [6] and its TDB store, a high-performance ontology repository able to manage large numbers of assertions in an efficient manner. OCP offers a subscription mechanism to notify about new arrivals of events, which is used by the Context Aggregator Agent.

Regarding the multi-agent architecture, it has been developed using JADE[1], a well-known agent framework that fosters the creation of agents and communication tasks. All the agents use Jena ontology models kept in main memory to store the assertions about the current context. Agents' rules are written in the Jena rule syntax (see [6] for details) and they are evaluated using the Rete rule engine available in Jena, whereas FSMs are represented by means of JADE FSM behaviors. Besides, agents CAA and AssesA are equipped with the OWL DL reasoner Pellet [12] to detect conflicting contexts by validating them according to Context_Ont (the use of an OWL DL reasoner is necessary since the Jena rule engine is not oriented to validate ontologies).

The particular multi-agent architecture deployed for the DIA project is composed as follows. Apart from CAA, we have developed an LCA in the Simple Context Agents group with the two rules R_{LC1} and R_{LC2} given in Section 4.2.2. The Complex Context Agents group includes two PatternCtx agents, one for the context `RelaxingSofa` containing the rule R_{RxSofa}, and the other one for the context `Sleeping` represented through an FSM. Moreover, in this group there is an agent for entailing the context `NoMovCtx` according to the rule R_{NM}. Finally, AssesA keeps the US rule R_{US1} given in Section 4.2.3 (there are no alarm sensors for the unattended events used in R_{US2}) together with the safe sleeping context rule R_{SS1}, and a rule R_{SS3} analogous to the latter for entailing an excessive lack of movement as safe if the occupant is in a normal-time relaxing-sofa context. These two SS rules are treated as exceptions to R_{US1}.

5.2 Results

Our AAL system has been evaluated with events logs produced through the DIA pilot study. These tests were used to measure two results: the *reaction time* of the system from the arrival of sensor data to the classification of the generated context as safe/unsafe, and the *reliability* of the system with respect to the number of false positives and false negatives. We explain these in more detail in the next two sections.

5.2.1 Reaction time

The reaction time to assess a new context generated by the arrival of sensor data is divided into three blocks: the time T_{OCP} required to create the corresponding event from the received sensor data, the time T_{COM} elapsed in communication among agents, and the time T_{CTX} needed to process the context. As a result, the reaction time T_R is calculated as:

$$T_R = T_{OCP} + T_{COM} + T_{CTX} \quad (1)$$

Note that our aim is not to obtain an exact approximation of T_R but an empirical checking about the interactivity level of the system. Thus, we only try to find the order of response time for our particular AAL system with a specific hardware configuration. In particular, we have used a single Ubuntu Linux 9.04 computer with a 32-bits Intel Core 2 Duo CPU at 2.10GHz and 3 GB of RAM to run the components of the AAL system, i.e., OCP and the entire agent network.

The first block is managed by OCP, which receives data from sensor adapters and transforms them in events according to Event_Ont. To this end, OCP queries SmartHome_Ont to obtain information about the type, location and element associated to the corresponding sensor. Thanks to the Jena TDB store used in OCP, the time required for executing queries is up to nearly 500 queries per second for ontologies with one million assertions. For our SmartHome_Ont containing several thousand assertions, the average time T_{OCP} required for querying it and creating an event is 3ms.

The agent communication time amounts to the sum of the communication times among CAA and Simple Context agents, CAA and Complex Context agents, and CAA and AssesA. Notice that CAA communicates with Simple and Complex Context agents in parallel. The transmitted data between pairs of agents are 180 assertions approximately ($\sim$4KB), divided into $\sim$100 assertions to represent the exchanged context and its associated events, and the rest of them to represent the arguments created to support the location information, activity patterns, etc. inferred about that context. Assuming that there is no delay in the communication channels, the average time required for each agent communication is 18ms. Therefore, $T_{COM} = 18 \times 3 = 54$ms.

Finally, the time needed to process a context is divided into the time involved in its creation, its evaluation against agents' rules, and validation of the information entailed by such rules. On one hand, the time required for rule evaluation depends basically on the number of rules and assertions against which the rules are evaluated. On the other hand,

[1] JADE is available at `http://jade.tilab.com/`

the time required for the validation process depends on the complexity of the expressions used to define concepts and relationships in Context_Ont and the number of assertions to be validated. The DL expressiveness level of Context_Ont is $ALCHF(D)$, which mainly indicates the use of concept and relationship hierarchies along with the use of the disjoint operator among concepts and functional relationships. According to this expressiveness, the complexity of Context_Ont is decidable and placed in the PSpace-complete level.

In particular, the processing of a context is executed in the next order: Firstly, CAA creates the context when a new event is received as explained in Section 4.2.2. The average time obtained for this task is 51ms(t_1). Secondly, the Simple Context Agents evaluate the context against their rules. In this case, the average time obtained for the evaluation of the two LCA's rules against ∼100 assertions –the number of assertions necessary to represent a context and its associated events– is 1.6ms (t_2). Then, the LCA's entailments are validated by the CAA in order to detect conflicting locations. This validation process consumes an average of 2ms (t_3). Next, the Complex Context Agents evaluate the context against their rules. The average times obtained for the RelaxingSofa-PatternCtx and NoMovCtx agents are 1.75ms and 1.29ms, respectively. Observe that both agents perform their rule evaluations in parallel and only the worst evaluation time, 1.75ms (t_4), is taken into account. Eventually, AssesA evaluates their three assessment rules, which require 5.4ms (t_5) in the average case, and validates the entailments of these rules, again consuming 2ms (t_6). It is important to mention that the argumentation process performed in CAA and AssesA requires a negligible time, since it only consists of comparing their premises or rules and selecting the most relevant one. Following these calculations, $T_{CTX} = t_1 + t_2 + t_3 + t_4 + t_5 + t_6 = 63.75$ms in the average case.

As a result of the previous calculations, equation (1) amounts to $T_R = 3 + 54 + 63.75 = 120.75$ms, which is the average reaction time of the AAL system deployed in the DIA project. With respect to scalability issues for this reaction time, the maximum number of assertions to represent more types of events and context information is expected to be not higher than one thousand assertions. This value may result in a slight increment for the partial T_{COM} and T_{CTX} times that should not affect the real-time reaction needed in the system. Likewise, the addition of more Simple Context and Complex Context agents (Fig. 3, boxes B and C) to manage new context information should not affect the reaction time in a significant way, since their inference process are performed in parallel with the rest of the agents.

5.2.2 Reliability

The reliability of our AAL system is determined by the number of false negatives and false positives generated by it.

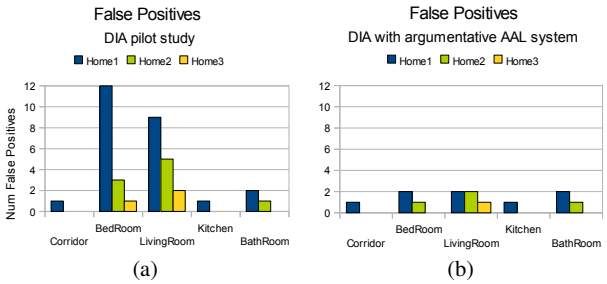

Fig. 4 False positives detected in the initial pilot study (a) and using the AAL argumentative multi-agent architecture (b).

Hence, a false negative arises when the occupant is actually suffering a problem that can be detected by an excessive lack of movement, but an unsafe context is not inferred by the system. Contrarily, a false positive occurs when an unsafe context is detected by the system because of an excessive lack of movement in the occupant's location, but in fact the occupant is not suffering any problem. Below we explain how our system handles false positives/negatives.

False negatives are avoided by the use of timers, since all the situations with an excessive lack of movement are eventually detected when the corresponding timer reaches zero. In the pilot study, a different timer was set depending on the occupant's location and daytime, and they were automatically and periodically adjusted according to the occupant's movement frequency (see [5] for details). This approach has been adopted in our AAL system by using those adjustable timers to monitor the `NoMovCtx` context. Hence, an argument claiming such a context as unsafe is created when its normal time is exceeded. If this situation happens when the occupant is in the `Sleeping` or `RelaxingSofa` context, it is also claimed as unsafe when the timer for that context reaches zero.

False positives were detected in the DIA pilot study and Fig. 4(a) shows statistics of the cases detected. As can be observed, most of false positives in the three houses were generated in the bedroom and living room. In particular, all the cases in bedrooms were detected when the occupants were actually sleeping, whereas practically all of those in the living room were detected when the occupants were resting in the sofa. These cases are due to the fact that the adjustable timers used in the study for detecting lack of movement do not take into account the usual hours for sleep/rest. Therefore, when an occupant has periods of high frequency of movement in such situations, the timers are set to low values which are surpassed when the occupant is in more normal periods of less frequency of movement. The Home1 occupant, see Fig. 4(a), is a good example of this case.

When evaluating the same situations in our AAL system, the sleeping/resting false positives now result in the occurrence of conflicts during context assessment. These conflicts

arise when the adjustable timer for the context `NoMovCtx` indicates that the normal lack-of-movement time has been exceeded, and therefore an argument claiming this situation as unsafe is generated through R_{US1}. However, if the timers introduced in the AAL system to control the duration of the `Sleeping` and `RelaxingSofa` contexts indicate a normal behavior for such contexts, the situation is also supported as safe by arguments generated through R_{SS1} and R_{SS3}. All the conflicts generated in this manner are resolved by establishing the situation as safe since arguments based on R_{SS1} and R_{SS3} are evaluated as more prevalent than arguments based on R_{US1} (note that R_{SS1} and R_{SS3} are exceptions to R_{US1}).

As a result of the argumentation processes executed during sleeping/resting activities, the number of false positives in the bedroom and living room is significantly reduced with respect to the pilot study as observed in Fig. 4(b), specially for Home1. Those that still appear in these locations are due to the normal times established for such activities has been exceeded. The rest of false positives in other locations, e.g., due to excessive lack of movement in the bathroom, are still occurring in our AAL system since the timers for those locations are the same ones employed in the pilot study. To avoid them, the usage of new sensors and the definition of pattern contexts are being studied.

6 Conclusion and Future Work

The development of Ambient Assisted Living (AAL) systems is one of the most active research lines within the Ambient Intelligence community. A fundamental service expected from such systems is an increase in safety for the occupant of a house. To this end, AAL systems rely on Intelligent Environments infrastructures to collect information about the situations that are taking place in the house. Although industry has made good progresses in the production of these infrastructures, the data provided by them often lead the AAL system to ambiguous and/or inconsistent pictures of the observed situations.

In this paper we propose the design of an AAL system which deals with such ambiguous and inconsistent context information by means of a qualitative approach. This approach is based on a multi-agent architecture where each agent supports its point of view about the occupant's context through arguments. Such arguments enable the development of a well-structured and sound reasoning process when inconsistent contexts are detected. In those cases, the evaluation of arguments through this reasoning process determines which context is the most plausible. As the evaluation of arguments relies on qualitative well-known criteria, it offers an alternative which is easier to understand and validate than quantitative approaches focused on intrinsic algorithms or complex models. Our proposal has been deployed and evaluated within a real AAL project and the results obtained show encouraging results in this direction. An evaluation of the AAL system in homes with a larger number of different types of sensors is being also developed at the moment.

We are currently exploring the inclusion of learning capabilities in the AAL system by means of data mining techniques to automatically extract pattern contexts from the occupant's habits. Another planned extension to the system is the addition of more structured representations of spatio-temporal knowledge and their associated reasoning modules to inform the argumentation process.

Acknowledgements This work has been supported by the Research Projects CARONTE (TSI-020302-2010-129) and DIA++ (TRA2009-0141), by the Fundación Séneca within the Program "Generación del Conocimiento Científico de Excelencia" (04552/GERM/06), Murcia, Spain, and by the Spanish Ministerio de Ciencia e Innovación under the FPU grant AP2006-4154.

References

1. Atallah, L., Yang, G.Z.: The use of pervasive sensing for behaviour profiling – a survey. Pervasive and Mobile Computing **5**(5), 447 – 464 (2009)
2. Bamis, A., Lymberopoulos, D., Teixeira, T., Savvides, A.: The BehaviorScope framework for enabling ambient assisted living. Personal and Ubiquitous Computing **14**, 473–487 (2010)
3. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American **284**(5), 34–43 (2001)
4. Besnard, P., Hunter, A.: Elements of Argumentation. The MIT Press (2008)
5. Botía, J.A., Villa, A., Palma, J.T., Pérez, D., Iborra, E.: Detecting domestic problems of elderly people: simple and unobstrusive sensors to generate the context of the attended. In: First International Workshop on Ambient Assisted Living, (IWAAL) (2009,LNCS 5518)
6. Carroll, J.J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K.: Jena: implementing the Semantic Web recommendations. In: WWW Alt. '04: Proceedings of the 13th international World Wide Web conference, pp. 74–83. ACM Press, New York, NY, USA (2004)
7. Cook, D.J., Augusto, J.C., Jakkula, V.R.: Ambient Intelligence: applications in society and opportunities for AI. Pervasive and Mobile Computing **5**, 277–298 (2009)
8. van Harmelen, F., McGuinness, D.L. (eds.): OWL Web Ontology Language Overview. World Wide Web Consortium (W3C) Recommendation (2004)
9. Muñoz, A., Botía, J.A.: ASBO: Argumentation System Based on Ontologies. In: M. Klusch, M. Pechoucek, A. Polleres (eds.) Cooperative Information Agents XII, *Lecture Notes in Artificial Intelligence*, vol. 5180, pp. 191–205. Springer (2008)
10. Nieto, I., Botía, J.A., Gómez-Skarmeta, A.F.: Information and Hybrid Architecture Model of the OCP Contextual Information Management System. Journal of Universal Computer Science **12**(3), 357–366 (2006)
11. Parsons, S.: Qualitative Methods for Reasoning Under Uncertainty. The MIT Press (2001)
12. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. Web Semantics: Science, Services and Agents on the World Wide Web **5**(2), 51–53 (2007)