Journal of Universal Computer Science, vol. 14, no. 9 (2008), 1529-1544
submitted: 30/4/07, accepted: 19/2/08, appeared: 1/5/08

Easing the Smart Home: Semi-automatic Adaptation in Perceptive Environments

Manuel García-Herranz
(Escuela Politécnica Superior, Universidad Autónoma de Madrid, Spain
manuel.garciaherranz@uam.es)

Pablo A. Haya
(Escuela Politécnica Superior, Universidad Autónoma de Madrid, Spain
pablo.haya@uam.es)

Abraham Esquivel
(Escuela Politécnica Superior, Universidad Autónoma de Madrid, Spain
abraham.esquivel@estudiante.uam.es)

Germán Montoro
(Escuela Politécnica Superior, Universidad Autónoma de Madrid, Spain
german.montoro@uam.es)

Xavier Alamán
(Escuela Politécnica Superior, Universidad Autónoma de Madrid, Spain
xavier.alaman@uam.es)

Abstract: This paper analyses the requirements of automation and adaptation in the so called perceptive environments. These environments are places with the ability of perceiving the context through sensors and other mechanisms. Focusing on personal/home environments, we present a first approach and prototype to semi-automatic adaptation of Perceptive Environments through a system of rule-based, configurable and modular agents, which are able to explain their behaviors and to adapt to the changing habits of the users. This prototype has been implemented over a real environment: a living room equipped with ambient intelligence capabilities. The core of the system relies on a set of modular agents equipped with rules. Those rules are composed of triggers, conditions and actions that enable them to express desired behaviors of the environment as well as to infer high-level context from low level context. One of the main objectives of the system is to leverage the control of the user over his/her own environment, making it easy to create powerful and personal behaviors without expert assistance. In this sense this work follows Greenberg's thought of making "simple ideas simple to be done" [Greenberg 07].

Keywords: ubiquitous computing, ambient intelligence, smart home, automatic adaptation, natural programming
Categories: H.1.2, H.5.2, I.2.5, C.2.4

1 Introduction

Since Mark Weiser coined the term *Ubiquitous Computing* in 1991 [Weiser 91], many problems and opportunities have arisen from that vision of a world rich in information and interaction, and many projects have been born in the intent of bringing that world of possibilities to reality.

One of the challenges arising from this vision is how to transform these *Perceptive Environments*, which are able to grasp their surroundings, into *Interactive Environments*, in which the user is able to communicate and interact with his/her living space, or into *Intelligent Environments*, in which the environment is able to take decisions based on the perceived context. In other words the question would be how to populate the environment with *context-aware* applications in the benefit of its inhabitants? The benefit of the inhabitant is highly dependent on his/her identity and type of environment. One example is that of an old woman at home compared to a young man at work; both share little but the desire for comfort and happiness, ambiguous terms with different meanings for each person. But does the inhabitant have this ability? We can say that everybody knows what they want and what they do not, but knowing and describing, as in the genie in the bottle tale, are different things. Hence the adaptation to the user's preferences becomes a non-trivial task.

Some projects, such as The Neural Network House, termed ACHE [Mozer 95], have taken an approach which does not explicitly ask the user for his requirements but rather infers them directly from his/her behavior. This enables the house to *"program itself"* [Mozer 98] for the user's benefit. To do this, ACHE *"monitors the environment, observes the actions taken by occupants and attempts to infer patterns in the environment that predict these actions"* using the advantages of **neural networks** [Mozer 98]. According to P. Maes [Maes 94] there are two main challenges that need to be solved by the software agents: **competence** and **trust**. Although neural networks may achieve a high degree of competence it is hard for a user to feel comfortable in delegating tasks to a system which is unable to explain its behavior.

In the Artificial Intelligence Lab at the MIT a reactive behavioral system, *ReBa* [Kulkarny 02], has been developed for the *Intelligent Room* human-computer interaction experiment, based on five design principles: *Context-awareness*, *conflict-resolution*, *adaptability*, *user-centricity* and *evolvability*. Kulkarny implemented the system as a set of rules bounded to an *activity*, which he termed an *activity bundle*. The system developers define activity bundles a priori by identifying the activity to which a reaction is relevant. If the user wants the system to react to a new activity he must install the appropriate bundle. ReBa provides a mechanism of macros that may be created by the user and invoked at any moment. This mechanism differentiates the "standard" reactions from the personalised ones in such a way that they will never merge, i.e. the personalised reactions will stay as patches over the standard system. Additionally, identifying the activity to which the desired reaction should be associated is not always a trivial task and is possibly far beyond what the user is willing to do. We believe that the system should allow the user to express his/her desires in a simple way and should integrate those desires with the rest of behaviors.

At Xerox Parc, one of the pioneering research centers in Ubiquitous Computing, they defined the PARCTAB system and developed four categories of Context-aware

Computing Applications [Schilit 94]. We will focus on the *context-triggered actions* applications, simple IF-THEN rules encoding a context that triggers an action. Two different applications have been implemented using these context-triggered actions: *Watchdog* and *Contextual Reminder*. The former allows the creation of rules over an *Active Badge* ("a tag that periodically broadcasts a unique identifier for the purpose of determining the location of the wearer" [Schilit 94]) of the type:

```
Badge location event-type action
```

where *badge*, *location* and *event-type* refer to the Active Badge; *event-type* can take one of the following values: *arriving*, *departing*, *settled-in*, *missing* or *attention*; and the *action* is realised as a Unix Shell command. An example for playing a sound message when coffee is ready would be:

```
Coffee Kitchen arriving "play -v 50 ~/sounds/ready.au"
```

On the other hand the *Contextual Reminder* allows popping up a message according to "when, where, who and what is with you" [Schilit 94]. One of the problems of this system is the language limitation for context definition. In the *Watchdog* application it is only possible to define the context of an Active Badge for a given time making it difficult to specify, for example, that the audio signal for the coffee maker should only be played if the TV is turned off. The same problem can be found in the *Contextual Reminder* application in which "when, where, who and what is with you" are not powerful enough to describe how things are, were or will be.

Aside from these problems, the rule-based system has proved to be effective for expressing desires in the form of reactions to context states. Some projects have used this mechanism to other uses. This is the case of *CONON*, an OWL encoded context ontology, developed at the *Institute for InfoComm Research* and the *School of Computing of the National University of Singapore.* In their work Wang et al. "*studied the use of logic reasoning to check the consistency of context information, and to reason over low-level, explicit context to derive high-level, implicit context*" [Wang 04]. CONON uses ontology reasoning rules, defined in OWL, to describe properties such as *Transitive* or *inverseOf.* Thus, if the property *location* is defined in the ontology as *Transitive*¸ knowing that Wang is located in the kitchen and the kitchen is located in the first floor, the system would reason that Wang is also located in the first floor; therefore the consistency of context information is acquired automatically. Additionally, in order to provide a mechanism for **extracting high-level context from low-level** information, the system allows the creation of *User-defined context reasoning rules* in a manner that can define a rule of the type: "If Wang is located in the bedroom and the bedroom light level is low and the bedroom drapes are closed then Wang is sleeping". We believe that this property of the rules is highly desirable.

Most critical in rule-based systems is the ability to define context. The lack of this descriptive power may cause inconvenience to the user due to unwanted or missing system reactions. We should mention three different projects: the *implicit Human Computer Interaction* project of the University of Karlsruhe in Germany [Schmidt 00], CONON [Tan 05] in Singapore and PRIMA [Brdiczka 05] in France. In [Tan 05] and [Schmidt 00] an extension is presented for defining context based in the concept of event. The former distinguishes between *Primitive events* (pre-defined in the system) and *Composite* ones (composed of primitive or composite events) and provide

a set of operators for defining sequences as well as periodic and non periodic events. In [Schmidt 00] an XML based language is specified to describe what was called the *implicit human computer interaction*. Using different types of semantics for grouping contextual variables, when all defined groups evaluate to true, the associated action is triggered.

These two projects extend the definition of context through sequences of events and grouping of variables. PRIMA defined a way to automatically identify situations in which the described context is ambiguous using situation splitting. This is highly useful for automatically adapting predefined rules, enabling to fine-tune the system to the user's desires.

Other related work includes the OCP system, developed at the University of Murcia [Nieto 06], *"a middleware which provides support for management of contextual information and merging of information from different sources"*. Based on Semantic Web technologies they developed a context inference mechanism based on rules such as do-if rules or do-for-all rules. The inference is carried out using the SWRL guidelines [Alesso 04] and the Jena platform as inference motor.

These works are indicative of the increasing interest of the scientific community in automatic reasoning over contextual information and demonstrate the growing usage of rule-based systems and its remarkable results and possibilities. Conversely none of these systems have fulfilled all the requirements for a user-centred and good-performing system required for everyday living spaces of non-technical users.

In the following section, based on the analysis of the projects presented above we will present the requirements that guided the development of our proposal.

2 Laying the foundations

Based on the definition of quality as *"a measure of adaptation to use"* and following the butler or personal assistant paradigm we pose the following question: what does the user expect from the highest-quality butler?

We believe, as other researchers [Hamill 06], that one of the main requirements of a butler (who we will refer to as *agent*) is to be **non-disturbing**. This could be seen as an idiosyncratic principle, more than as a requirement. Consequently, the agent should not bother the user more than necessary.

Additionally, the user may want to **express** his/her desires in a simple and natural way. Furthermore he/she may want to ask the agent for an **explanation** of its reactions, in order to understand and correct its behaviors. Finally, he/she will expect the agent to **learn** by itself and adapt to small changes, without being explicitly requested.

Following the principles of competence and trust stated by P. Maes [Maes 94] an agent should obtain competence by acquiring explicit knowledge from the user and implicit knowledge from autonomous learning; trust, on the other hand, will be gained through the ability to explain its behaviors in the user's language.

Since preferences may vary from user to user and through environments and situations it is also desirable to have a high degree of system **modularization**.

In summary, our system is based on two principles: **non-disturbing** and **modularization**, and over three requirements: **ease of expression**, **ability of explanation** and **automatic learning**.

3 Design principles

Since every kind of knowledge can be encoded in a language, with its strengths and weaknesses, our task becomes more like finding the language that best describes the needs of our world of preferences, desires and conclusions and allows us to meet our requirements of expression, explanation and learning. Expression and learning can be met by almost any language but the explanation requirement forces us to a language capable of describing its encoded knowledge in a human readable way, thus excluding black-box alternatives such as neural networks.

Furthermore it is necessary to define what kind of knowledge is going to be encoded in that language; in this way we focus on two different issues. First and most importantly, we want to express the desires and preferences of the user. Since most of these desires involve the user's will to having something done when some situation arises, the language should be capable of **describing** the "something to be done" and the "arising situations", in other words, **actions** and **context**. Secondly, regarding the conclusions, it is also desirable to find a language with the ability of **describing high-level context from low-level information**. Finally, the language should be human-readable. Those requirements have led us to the choice of a **rule-based** system.

One of the key issues of context-aware applications relates to when to check the context in order to act as expected. Because supervised environments can grow to considerable sizes and given that not every component has the same timing constraints, determining the time intervals for checking the state of the context becomes a challenging problem. Thus we have decided to take an **event-based** approach in which we assume that only a change in the environment can trigger another change on it.

4 A Language for Perceptive Environments

Following the design principles stated in the previous section we have developed a rule-based language for a modular agent system. In this manner the system will be composed of independent agents, each of which comprises a set of rules encoding reactions to context states.

Each rule is composed of three parts: *triggers*, *conditions* and a single *action*:

- *Triggers*: supervised context variables responsible of activating the rule.
- *Conditions:* a set of pairs "context variable-value" representing a context state that needs to be satisfied for detonating the action.
- *Action:* a pair "context variable-value" to be set when in a triggered action all its conditions evaluate to true.

Most systems only differentiate between conditions and actions, using the former as triggers. Thus, behaviors can be encoded as ***"If <conditions> then <action>"*** We have observed that in many situations not every condition should trigger the action.

Adding triggers to the structure allows expressing behaviors of the type ***"When <triggers>, if <conditions>, then <action>"***

As an example for the need of triggers let's imagine a living room with a TV and a dimmer light with three values: HIGH, LOW and OFF; in this scenario, when turning on the TV, the user wants to set the light to its LOW value if it was in the HIGH value. This behavior can be encoded into the context *TV-ON AND LIGHT-HIGH* with the associated action *LIGHT-LOW*. If no triggers are specified any change on any variable of the conditions will cause the revaluation of the rule. Thus, if the user sets the light level to HIGH when the TV is ON then all the conditions will evaluate to true and the system will set the light level to LOW, against the user's will. In other words, *"If TV = ON and LIGHT = HIGH then LIGHT = LOW"* is not powerful enough to express *"When TV = ON if LIGHT = HIGH then LIGHT = LOW"*

Additionally, behaviors of the type ***"While… if… then…"*** can also be expressed using rules to activate/deactivate agents that, *while* active, will apply their rules. More precisely, it is possible to express behaviors of the type ***"While <agent is active> when <triggers > if < conditions> then <action>"***

Although the user may want to specify more than one action per behavior we have chosen a system in which each rule only supports a single action. More actions per rule can be encoded using several rules with a single action each. This mechanism simplifies the task of identifying wrong rules in the learning process.

5 Architecture

The system is based on a modular architecture in which a set of rules –with a common goal– are grouped under an independent unit. We will refer to that unit as an agent. Agents are represented in the Blackboard-based world representation [Haya et al, 2004] as virtual entities at the same level as persons, locations and devices.

As best-practice, rules grouped under the same agent should share an object –the owner of the agent, for whom they work– and a subject –the goal they chase.

Intelligent Environments possess two characteristics that make them especially unique: their **changing nature** and their **human population**.

5.1 Dealing with changing context

Human populated environments are subject to constant changes. These changes can be divided into long-term and short-term.

Long-term changes: Simple evolution

By long-term changes we refer to those modifying the structure or population of the world i.e. adding a new light, removing an old one, a new person joining the community or a new intelligent space added to the world.

The environment's behaviors can be updated through the addition or removal of agents without affecting the rest of the system. If an agent is in charge of inferring the location of the inhabitants and a new, more reliable, mechanism is installed, removing the old agent will affect no other part of the system. Since every reasoning unit is independent, the only change will be more accurate location information.

Additionally, if a coffeemaker is added to the kitchen, an agent could be created to make coffee at 7:00 a.m. from Monday to Friday or to announce when coffee is ready. If the coffeemaker is removed, so will the agent.

Short-term changes: Dynamic adaptation

While the structure or population of the world remains unchanged, some set of preferences may depend on local variations. Thus, even if Xavier exists and there are lighting devices in his house, it does not make sense to apply his preferences when he is not at home. Combined with the agent's representation as another element of the world, the modular architecture allows the creation of "meta-agents" in charge of activating/deactivating other agents according to the perceived context.

5.2 Dealing with human population

Computing technologies have been applied to many human-populated places but personal environments present a main difference over all of them: they are personal. When talking about hospitals or factories the inhabitants participate on achieving an external goal – i.e. to reduce treatment time or to increase productivity – and they are externally rewarded to deal with the burden associated to achieving that goal. On the other hand, the only goal of a personal environment is the inhabitants' comfort, towards which every burden is a step backwards.

Computing technologies, when applied to personal environments, must deal in the most sensible way with human-nature if they are to succeed.

Hierarchical structures: Spreading responsibilities

Human beings, due to their social nature, are used to deal with hierarchical structures in which tasks and responsibilities are spread among their members. Within the house this structure is historically easy to see: gardeners, housekeepers, butlers and so on [Hamill and Harper, 2006].

Going back to computing technologies, a modular architecture benefits the imitation of these structures, bringing two main benefits: **emulating well-known human structures** and **spreading responsibilities**.

The former releases the user from learning new paradigms of task distribution. Dealing with an already known structure helps the user to feel more familiar and comfortable. The latter, on the other hand, has a direct impact on the amount of trust the user places on the system –one of the main problems to be solved as stated by P. Maes [Maes 94].

While in a centralized system the failure of a part is translated into a loss of confidence over the whole, in a modular architecture the responsibility relies only on the failing module: Wherever the failure on the lighting preferences module is, the good performance of the security module – and the trust it receives from the user – will not be affected.

It should be noted that the Blackboard system of Interact [Haya 04], over which the work described in this paper is implemented, already deals with collisions of actions from different agents over a single entity. Thus, the agent will only have to manage its collisions. This is much simpler to do and, since agents are associated with users, can be done dealing with user's preferences.

Specific reasoning: Simplifying definition

Divide et impera[1]. It is natural for us to attack simple and specific problems and, when dealing with big and complicated ones, to split them into small parts.

When talking about a complex task such as configuring our home to adapt to our wide range of preferences, a modular architecture allows us to attack the problem in a limited, small scale, for instance configuring the TV, the lights and other preferences over a well-defined scenario. These small and individual tasks are easier to handle than the overwhelming goal of "configuring the home". In this way, the "configuration of the home" is just the result of the combined actions of the "configuration of the parts" represented in the different agents created to deal with the smaller problems.

We believe that simplifying the language and allowing the decomposition of problems are keystones to simplicity in adaptation. As an example, we can mention the work of one of our first year CS students: using the rules/agents system and with no prior programming knowledge, he created some agents for different purposes such as augmenting a phone (lowering all volumes in the room when somebody is talking on the phone), defining scenarios and their associated behaviors (such as watching TV, having siesta, working or relaxing) or augmenting a sofa to make it an activity identifier interface. All the agents work as a whole –i.e. defining in the sofa that a siesta is taking place triggers the siesta scenario– but each was developed in a gradual process of simple steps.

5.3 Reusability

A modular architecture opens up a natural path to reusability: behaviors can be easily exported from one place to another i.e. "I like the way you control the lights on your living room, can I copy it?" When grouping behaviors with a common goal, over a unique agent, the only remaining task to achieve reusability is to define the isomorphism between the two environments i.e. "what you call *lamp 1* in your living room is called *main lamp* in mine, your *radio* is my *radio 1*…". Thus, if a user wants to export some set of behaviors from a place to another he only has to copy the desired agent and establish the correspondence between the elements of the two environments. This idea is similar to that of the *Digital Recipes* explained by Newman, Smith and Schilit [Newman 06]

6 Easy Automation

At this point it is possible to see that most design decisions, from the simplicity of the language to the modular architecture capabilities or the rule-based/event-based structure aim towards the same goal: to make the process of environment automation as **easy** and **natural** as possible.

Natural in the sense stated by Myers et al [Myers 04] as "faithfully representing nature or life" meaning that "it works in the way people expect". When defining programming as "the process of transforming a mental plan in familiar terms into one

[1] Divide and rule

compatible with the computer" [Hoc 90] Myers et al declared that "The closer the language is to the programmer's original plan, the easier this refinement process will be" [Myers 04]. By language we understand not only the language itself but also the pieces in which it is contained, meaning, the agents. In the same article Myers et al. observe through two studies examining "the languages and structures that children and adults naturally use in solving problems", that "an event-based or rule-based structure was often used, where actions were taken in response to events".

Easy, on the other hand, in the sense stated by Greenberg [Greenberg 07] as "removing low-level implementation" so "programmers can rapidly generate and test new ideas, replicate and refine ideas, and create demonstrations for others to try". In the same work he demonstrated, in relation to groupware application development, how a good tool for developing can break the bottleneck suffered in Gaines' BRETAM - a phenomenological model of how science technology develops over time - [Gaines 99] (see Figure 1). We believe this situation is quite similar to the one encountered in the Intelligent Environments applications development.

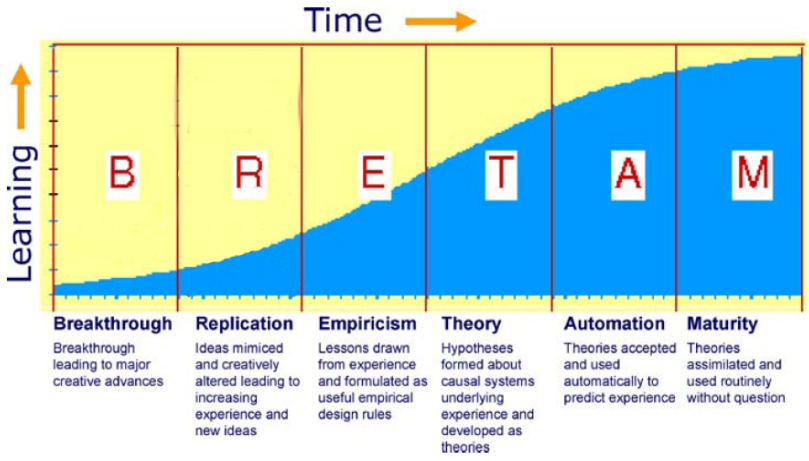

Figure 1: BRETAM science technology development model. From [Gaines 99]

7 Prototype

Dealing with the three requirements stated in section 2: ease of expression, ability of **explanation** and **automatic learning**, we have implemented a first prototype of the agent system.

The first requirement is achieved through the almost-natural language of rules described in section 4. This language uses some special characters to differentiate between trigger, conditions and action (see Code Section 1). Each trigger can be a change on a property, on a relation or on the existence of an entity; the conditions are of the form *element operator element* where an element can be an *entity*, a *property* of an entity, a *relation*, or a mere *value*, and the operator can be *equal*, *not-equal*, *greater*, *smaller*, *exists* or *not-exists*. Not every operator can be applied to every type

of element. Finally, the actions are composed of an *element*, an *operation* and another *element.* The operation could be *assign*, *assign-opposite-value*, *minus*, *add*, *create* or *delete.* As with the conditions the agent will check for correct action syntax.

```
tv:tv1:status     ::     tv:tv1:status     =     ON     ;
dimlight:lampv1:value   >   20                          ->
dimlight:lampv1:value := 20
```

Code Section 1: Rule following the template *Triggers :: conditions -> action.* "Whenever the TV status changes, if the TV is ON and the lamp is set to HIGH then lower it to MEDIUM".

In this first approach the explanation requirement is accomplished through a mechanism of on-request traces by which the system shows its internal process of learning, triggering, evaluating and reacting as the relation between context changes and its rule inference process (see Code Section 2):

```
- Changed property: tv:tv1:status to value: OFF
Triggered rules:
      - [ ] Rule 20: tv:tv1:status :: tv:tv1:status = ON
(false)-> light:lamp_1:status := OFF (10)
      - [ ] Rule 21: tv:tv1:status :: tv:tv1:status = OFF
(true)-> light:lamp_1:status := ON (10)
Applied rules:
      - [ ] Rule 21: tv:tv1:status :: tv:tv1:status = OFF
-> light:lamp_1:status := ON (10)
```

Code Section 2: example of the explanation mechanism. The header corresponds to the change responsible for the process (TV *status* changing to OFF) then a list of all triggered rules (all rules with that property within its triggers). Finally, a list with the applied rules (those whose conditions evaluate to true). Every rule begins with a checkbox determining if it is activated ([ ]) or deactivated ([X]) and its rule number -i.e. Rule 21.

This mechanism allows the user to understand the whys and wherefores of the system's actions and to pinpoint possible failures. Maybe there is an invalid rule running, or some rule the user does not want any more i.e. "I don't want you to turn the lights off anymore when I'm watching the TV" – or it could be a rule with an incompletely defined context – i.e. "Yes, I want you to turn off the lights when I'm watching the TV but... only if there is no one reading in the room" – or maybe it is the perceived context what is wrong – i.e. "You thought I was having siesta, and you turned off the door bell but I'm not having siesta" – shifting the responsibility away from the agent and into the context supplier (which can be another agent that can be required to explain its actions)

This mechanism has been extremely useful in the development and implementation of the system. When actions are tagged with their executor's id, it generates a responsibility chain i.e. "Who turned off the lights?" "I did" "Why?" "Because you told me to turn them off when you take a siesta…" "Wait, who said I was taking a siesta?" "I did" "Why?" … – thus permitting to follow the flows of information and reasoning until finding the origin of trouble.

Since users can constantly supervise the behaviors and internal processes of the whole system, they never get the feeling of losing control.

Regarding the learning process, every rule has an associated weight reflecting its degree of confidence. Every time a rule is executed the system marks it for feedback training; if within a fixed length of time (currently one minute) no other process contradicts its action the rule is positively rewarded by incrementing its weight, otherwise the value is decremented.

Finally, all these processes are chained into the following algorithm:

```
For every change in the context
Educate all rules
Update the true-false value of the conditions
       For every triggered rule
              If its conditions evaluate all to true
                     Send its actions to the Blackboard
```

Code Section 3: Algorithm for execution of rules according to a context change

As seen in Code Section 3, conditions are updated as the context changes, spreading the computational load over time and speeding up trigger checks. This response time is a critical issue: a delay of one second in turning on the lights can ruin for the success of the application. Additionally, the extra communication of this alternative grows to zero as the number of rules is increased.

One of the main questions that naturally arise at this point is "Why not use existing algorithms for pattern matching and production rule systems?" Rete [Forgy 82] is a very efficient algorithm for comparing large collections of patterns and objects, chaining inferences in a fast and low-cost manner. Additionally some rule languages such as CLIPS or JESS already support Rete, indeed, one of the most popular in inference engines, including some UBICOMP systems such as [Zhand 04] or [Hall 01]. Two problems wrap this algorithm when applying it to our research: centralization and certainty. Rete can be used in systems "containing from a few hundred to more than a thousand patterns and objects" [Forgy 82] but in our system patterns are distributed along different agents each of which must only deal with its own small set of rules. In other words, instead of a huge expert system, ours can be seen as the cohabitation of many little ones. Additionally, one of the main benefits of Rete is the fast inference chaining i.e. "If, from A, we can infer B and from B we can infer C then from A we can infer C"– As stated previously, the collision solving mechanism is down in the Blackboard layer, thus any command on a context variable –"Turn on the light"– represents only a will, e.g. The blackboard, through its priority mechanism, is in charge of transforming that will into a real action, ignoring it otherwise. Consequently, no inference chaining is possible –i.e. "If, from A, we can infer B and from B we can infer C, we cannot infer C from A until B is acknowledged by the blackboard". In other words, agents can only make one-step inferences. In conclusion, Rete-type algorithms are unsuitable for distributed one-step inference engines such as ours.

8 Case study

This prototype has been implemented in a real environment (a living room and an office space, see Figure 2) and has allowed replacing multiple ad-hoc applications as well as to develop new and varied in a simple, comprehensible and fast manner with very satisfactory results. At the time of this writing these applications are distributed within twelve different agents of different kinds. We present some of them with a selection of their rules written in plain English but following the *triggers (when), conditions (if), actions (then)* structure:

- security agents, to allow or deny the entrance to the lab
  When the Door_card-reader reads a card, if the read_card is Manuel:card and the door is closed then open the door
- high-level context inferrers, such as the location supplier
  When the Door_card-reader reads a card, if the read_card is Manuel:card and Manuel is not located in lab_B403 then Manuel is located in lab_B403
- device controllers, for enhancing the light switch or the telephone
  When the switch_up is pressed, if the main_light is turned on and the secondary_light is turned off then turn on the secondary light
- scenario agents, which define scenarios such as *siesta* or *Manuel relax* [García-Herranz 07]
- meta-agents to activate/deactivated other agents, for shifting the behavior of the light switch or changing the "definition of relax" according to the inhabitants
  When Manuel:location changes, if Manuel is not located in lab_B403 then deactivate agent_Manuel_siesta
  When the Switch_RF-reader reads an RF-id, if the read_RF-id is Manuel:RF-id then deactivate agent_Switch and activate agent_Manuel_Switch
- agents to control an interactive display [García-Herranz 07]
- or the most common and natural ones, those in charge of the plain preferences, such as lighting or vocal messaging.
  When Manuel:location changes, if Manuel is located in lab_B403 then say on B403_synthesizer "Hello Manuel"

It should be noted that these agents have been programmed not only by engineers but also by non programmers [García-Herranz 07] following their own intent, preferences and ideas. For doing so they were provided with a GUI to create rules in a natural way by dragging and dropping context elements into three boxes –i.e. "When" "If" "Then".

Figure 2: Snapshot of the living room and details of some system parts and components: EIB and USB Phidget elements, IP camera, controlled and augmented objects such as an interactive couch, telephone, table or coffeemaker and detail of the camera-microphone of the augmented table. (Main picture with © Diego Sinova)

9 Conclusions

We should point out first the **use of triggers in the rule language**. This simple and intuitive concept has been a keystone in developing the system as well as in defining context over this type of environments. Secondly we must remark the **wide possibilities brought by the modularization** of agents together with their representation as another part of the world.

In relation with the challenge of learning automation we have found that a semi-automatic approach in which learning is based on knowledge and restricted domains explicitly stated by the user is much simpler, effective and less annoying for him than those in which the environment learns "as much as possible". **We believe that the path of automation in Perceptive Environments should go through enhancing human control, not over diminishing it**. Consequently, future systems will require means for naturalizing and extending their control mechanisms, helping the user to know and express his/her desires. In this sense, learning should play a tuning and suggesting role in the system, "an aid to deal with the genie in the bottle".

10 Future work

This first prototype has served as a basis to contrast our hypothesis over a real environment; however, it is still limited. Besides the implementation of a GUI to easy the definition of rules (towards the completion of the **expression** requirement) we are currently working in the construction of an **ontology** to define the different elements of the reasoning world. On one hand, the ontology-defined system will allow the

addition of new reasoning elements to the language. On the other hand, the linguistic and procedural information will be encapsulated in each element of the ontology, allowing agents to **dynamically "understand" the language** as well as implementing a natural language solution for the **explanation** requirement, similar to that of [Montoro 04], able to take the linguistic information directly from the ontology with no need of an external corpus.

Having developed a GUI to deal with the **expression** requirement we are currently working in the other two requirements: **explanation**, through a mechanism of queries were users cannot only trace the internal reasoning of agents but also ask them for specific explanations, and **learning**, studying the oscillations in the confidence factor to identify misbehaving rules.

Agents are intended to deal with home environments (personal agents in environments with few inhabitants and not too many visitors). However, its scalability to larger contexts such as public infrastructures should be studied.

Regarding the internal structure of agents, we believe that integrating meta-agent capabilities into agents will clarify and ease the process of creation. Thus every agent should have, besides the actual group of rules, another three groups: *on activation* rules (to apply whenever the agent becomes active) *on deactivation* rules (to apply whenever it becomes inactive) and *activation* rules (to auto activate the agent).

Acknowledgements

This work has been partially funded by the Spanish Ministry of Science and Education, (project TIN2004-03140) and by U.A.M-Grupo Santander (project Itech Calli), and is part of the UAM-SOLUZIONA AmI Laboratory research program. Special thanks to Eran Eden and Manuel Freire for their recommendations.

References

[Alesso 04] Alesso, H.P.: Smith, C.F. Developing Semantic Web Services. A K Peters, Ltd. (2004) ISBN 1568812124.

[Brdiczka 05] Brdiczka, O., Reignier, P. Crowley, L.J.: Supervised Learning of an Abstract Context Model for an Intelligent Environment, Smart Objects and Ambient Intelligence. SOC-EUSAI 2005 (Grenoble 2005)

[Forgy 82] Forgy, C.: Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. Artificial Intelligence 19 (1982) 17-37

[Gaines 99] Gaines, B.: Modeling and forecasting the information sciences. Inf Sci 57/58: (1999) 13-22

[García-Herranz 07] García-Herranz, M., Haya, P.A., Alamán, X. Martín, P.: Easing the smart home: augmenting devices and defining scenarios. Second Intrenational Symposium on Ubiquitous Computing & Ambient Intelligence 2007 (UCAmI'07) September 2007, Zaragoza, Spain. Accepted.

[Greenberg 07] Greenberg, S.: Toolkits and Interface Creativity. Journal Multimedia Tools and Applications, 32(2), February (2007). Kluwer. Special Issue on Groupware. Available at SpringerLink.

[Hall 01] Hall, D., Le Gal, C., Martin, J., Chomat, O., Crowley, J.L.: MagicBoard: A contribution to an intelligent office environment. Robotics and Autonomous Systems. 3-4, 35, (2001) 211-220

[Hamill 06] Hamill, L., Harper, R.: Talking Intelligence A Historical and Conceptual Exploration of Speech-based Human-machine Interaction in Smart Homes. International Symposium on Intelligent Environments (Microsoft Research, Cambridge, United Kingdom, Apr 5-7, 2006), 121-127

[Haya 04] Haya, P.A., Montoro, G., Alamán, X.: A prototype of a context-based architecture for intelligent home environments. International Conference on Cooperative Information Systems (CoopIS 2004), Larnaca, Cyprus. October 25-29, 2004. 477-491.

[Hoc 90] Hoc, J.M., , Nguyen-Xuan, A.: Language semantics, mental models and analogy. Eds. Psychology of Programing. Academic Press. London (1990) 139-156

[Kulkarny 02] Kulkarny, A.A.: A reactive behavioral system for the intelligent room. Massachusetts Institute of Technology. (2002)

[Maes 94] Maes, P.: Agents that Reduce Work and Information Overload. Communications of the ACM, 7, 37, (1994) 31-40

[Mozer 95] Mozer, M.C., Dodier, R.H., Anderson M., Vidmar L.,, Cruickshank Iii, R.F., Miller, D.: The Neural Network House: An overview. Current trends in connectionism, L. Niklasson and M. Boden, Eds., Lawrence Erlbaum, Hillsdale, NJ, (1995) 371-380.

[Mozer 98] Mozer, M.C.: The neural network House: An environment that adapts to its inhabitants. In Proceedings of the AAAI Spring Symposium on Intelligent Environments (AAAI98). (1998)

[Montoro 04] Montoro, G., Alamán, X., Haya, P.A.: A plug and play spoken dialogue interface for smart environments. Fifth International Conference on Intelligent Text Processing and Computational Linguistics (CICLing'04). Seoul, Korea. February 15-21, 2004. 360-370. Lecture Notes in Computer Science (LNCS), volume number 2945: 360-370. ISSN 0302-9743

[Miers 04] Miers, B.A., Pane, J.F., Ko, A.: Natural programming languages and environments. Communications of the ACM 9, 47 (2004) 47-52

[Newman 06] Newman, M.W., Smith, T.F, Schilit, N.: Recipes for Digital Living. IEEE Computer, 2, 39, (2006) 104-106

[Nieto 06] 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, (2006) 357-366

[Schmidt, 2000] Schmidt, A.: Implicit Human Computer Interaction Through Context. Personal and Ubiquitous Computing. 2/3, 4 (2000)

[Schilit 94] Schilit, B., Adams, N., Want, R.: Context-Aware Computing Applications. Workshop on Mobile Computing Systems and Applications (IEEE, Santa Cruz, CA, US) (1994)

[Tan 05] Tan, G.J., Zhang, D., Wang, X., Cheng, S.H. (2005) Enhancing Semantic Spaces with Event-Driven Context Interpretation. In Proceedings of Pervasive Computing, Third International Conference. (PERVASIVE, Munich, Germany, May 8-13, 2005), 80-97

[Wang 04] Wang, H.X., Zhang, Q.D., Gu, T., Pung, P.H.: Ontology Based Context Modeling and Reasoning using OWL. In Proceedings of PerCom 2004 (Orlando, FL, USA, March) (2004) 18-22

[Weiser 91] Weiser, M.: The computer for the 21st century. Scientific American, 265, 3, (1991). 94-104

[Zhang 04] Zhang, T., Brügge, B.: Empowering the User to Build Smart Home Applications, ICOST 2004.