Journal of Universal Computer Science, vol. 16, no. 12 (2010), 1633-1649
submitted: 29/12/09, accepted: 5/5/10, appeared: 28/6/10 © *J.UCS*

Towards a Ubiquitous End–User Programming System for Smart Spaces

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

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

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

Abstract: This article presents a rule–based agent mechanism as the kernel of a ubiquitous end–user, UI–independent programming system. The underlying goal of our work is to allow end–users to control and program their environments in a uniform, application–independent way. The heterogeneity of environments, users and programming skills, as well as the coexistence of different users and domains of automation in the same environment are some of the main challenges analyzed. For doing so, we present our system and describe some of the real–environments, user studies and experiences we have had in the development process.

Key Words: Ubiquitous Computing, Human-centered computing, Rule-based processing, Command and control
Categories: H.1.2, I.2.5, I.2.4

1 Introduction

Many computers in the world run the same operating system and almost every one of them is based in the same screen–windows interaction paradigm. They are compact elements segregated of the environment, with an exiguous set of hardware, in which all their capabilities are within reach. Applications are designed with a defined purpose and users tend to use them just for it. Real spaces, on the other hand, are composed of a disaggregated, sizable set of elements, operated and managed by different people (preferences, needs and goals) and whose objects may be used in different ways.

The computational capabilities of intelligent environments extend their control possibilities from *direct control* to *indirect control*, in which the actions are not explicitly commanded or executed directly by the user but inferred from the on going context i.e. context–aware applications. While in the former the user is "on the loop" by definition, in the latter the inference process can be left up to the artificial intelligence of the environment [see see Mozer (1998) and Youngbloodet al. (2005)], commended to some third parties such as managers or developers [see Román et al.

(2002), Kulkarni (2002) and Schmidt (2000)] or brought closer to end–users, in an effort to put them back "on the loop".

As stated by Davidoff et al. [see Davidoff et al. (2006)], personal spaces such as the home *play a role in group and individual self–definition*: rather than just using them for a specific function, users pour their personalities and lives in the way they use and transform their personal environments. Therefore, allowing users to keep the control they used to have, as their environments become populated with computational elements must become a major concern.

In addition, this statement has a second implication when we look at independent living. In an aging society as ours more and more people are forced to trade their homes for institutions or to move to relatives' houses in order to get the assistance they need. While doing so results in loosing independence and a big part of what define themselves, dangerous situations may arise from not willing to: daily activities become more difficult, health issues and accidents harder to prevent and detect and, at the end, independence turns out into loneliness and risks. The dilemma between social alienation and proper care is common to elders and other collectives such as people with Down Syndrome trying to live and independent life. Thus, Ambient Intelligence is of especial importance to this collectives with special needs, relying on the assistance of others to help and supervise them in their daily living. Ambient Intelligence, should consider not only issues such as supporting daily activities or ambient diagnostics [see Cai and Klein-Seetharaman (2004)] but also closing the gap that separates individuals in modern societies as well as allowing them to extend the control they have over their environments (limited by their special conditions) reinforcing the role they play in their sef–definition.

This is the example of Alfred [see Gajos et al. (2002)] (designed for the IRoom), a multi–modal macro recorder allowing users to create automatic direct control structures that can be executed according to some hardware trigger (but with no possible conditionals) in a first step to end–user indirect control. Following this line, our work pursues an **application–independent** indirect control programming system combining the necessary **flexibility** to program complex behaviors with the **simplicity** required to allow novice users to program their environments.

2 Related work

Besides some projects such as *House_n* [see Intille and Larson (2003)] in which the system puts the user "on the loop" by giving information rather than by acting on her behalf, most systems have pursued a way for users to program their environments, exploring different kinds of easy programming paradigms. Thus most of them are based and focused on a particular UI.

MediaCubes [see Hague (2005)] is based on a script piping mechanism in which scripts are created and piped through a Tangible UI (TUI) based on cubes whose faces can be connected. While the keystone of the system is the *Do-when* cube, programming is seen like a chain of scripts rather than as rules.

CAMP [see Truong et al. (2004)] focus in programming capture applications such as "When Jim and Jane talk, record". This is done through a GUI in which English sentences are constructed using word tokens (i.e. Fridge Magnet Poetry metaphor).

Accord [see Rodden et al. (2004)], a jigsaw puzzle based TUI, emphasizes easy reconfiguration rather than programming while iCap [see Dey et al. (2006)] is a simple GUI to create if–then rules by drag and dropping elements into a matrix. While we share several design goals with iCap, we believe that this particular interface is best suited for novice users than for experience ones. In addition the lack of events as triggers limits the expressiveness of the language.

In contrast to those, Zhang and Brügge [see Zhang and Brügge (2004)], focus on a more flexible language, using JESS as the basis rule specification language to define ECA rules. Nevertheless, JESS syntax is not easy to understand and has not been designed to deal with end–user programming. Thus Zhang and Brügge's system relies on a *Rule Builder* and *Rule Debugger* to limit complexity and deal with end–user's limited programming skills.

In this work we pursue a system whose potentials can be compared to those designed for professional programmers such as Gaia [see Román et al. (2002)]. Contrary to JESS based solutions, we believe that the system must be end–user oriented from the underlying programming language and programming structure up to the UI. In such a language, flexibility and expressiveness is achieved incrementally, starting on a simple base language, designed to deal with novice users, extended with new concepts of self–contained complexity. In addition, the language should be designed to be UI–free, focusing on a natural reasoning structure rather than in a natural manipulation mechanism. Finally we pursue a domain–free system in which any element of the environment can be equally controlled and in which different users and sets of preferences can cohabit, allowing end–users to also create their coordination structures.

3 Controlling the environment

Indirect control can be classified into two subcategories: *preprogrammed indirect control* and *programmable indirect control*. The former refers to applications programmed by professionals to deal with specific tasks. Even though these applications may automatically adapt to some changes their goal and means are fixed. Providing parameters to allow end–users to personalize them has two associated drawbacks: personalization becomes **application–dependent** (i.e. each application has different parameters) and an **inverse relationship between flexibility and simplicity** arise (i.e. more flexibility implies more parameters).

These two drawbacks affect to what we believe is the core of a "disappearing technology": Naturalness and Easiness. In this sense, we understand naturalness as the way in which a system preserves the basic human structures and methods, and easiness as the manipulation capabilities it provides: i.e. preserving the *status–quo* while increasing the potential.

While these kind of preprogrammed applications are well suited to deal with common, transparent or complex problems they lack the flexibility to deal with most of the small personal preferences of user's daily life.

4 Indirect control

To deal with programmable indirect control without the application–dependent and adaptability–simplicity drawbacks, we propose a rule–based language as the kernel of every indirect control interface [see García-Herranz et al. (2008)].

In other words, every UI "speaks" the same kernel language so control structures can be created and accessed through any of them. From the end–user point of view, the underlying programming language remains invariable and UIs will be chosen just in terms of interaction capabilities: different UI would be preferred while driving a car (with less time, resources and attention) than while working on a PC but the mental plans will be the same. The closer the kernel language is to the end–user's original mental plan, the easier it would be the programming process and the more UI's designers can focus just on interface issues. Among others, Myers pointed at rule-based languages as the ones naturally used by users in solving problems [see Myers et al. (2004)]. For a more detailed discussion on the matter [see García-Herranz et al. (2008)].

4.1 Few natural concepts, many alternatives: Rule's Grammar

The system described in this article runs over the Blackboard middleware developed by Haya [see haya04prototype]. This middleware abstracts every physical or logical object into a high–level Blackboard *entity* (e.g. lamp_1), of a *type* (e.g. light), with some *properties* (e.g. status [*ON*/*OFF*]), and possible *relations* with other entities (e.g. located_at room:lounge).

Basically, our rule language describes actions associated to context. Special attention is paid to short–term effort and long–term restrictions (as in Paperts' ideal of "low–threshold no ceiling" [see Papert (1980)]). We believe, as Repenning and Ioannidou [see Repenning and Ioannidou (2006)] that "anxiety results if challenges outweigh the skills, while boredom results if skills outweigh the challenges".

In order to balance the language's description power and its simplicity, we based our design in two principles: **keeping a simple base language** and **isolating complexity** in the elements requiring it.

Regarding the simple base language, we chose a "photographic" representation of the world [see Haya et al. (2004)], in which the TV is either ON or OFF and no event is directly represented but indirectly inferred from the TV changing from ON "in one picture" to OFF in the next one. This forces the differentiation of *triggers* (e.g. the TV turning ON) and *conditions* (e.g. The TV turned ON) in the rule but preserves a unique and simple view of the world. Actions, on the other hand, just refer to either changing the state of the world (e.g. turning on a light), expanding it (i.e. creating a new agenda event) or restricting it. The only flexibility the base language provides is the use of "wildcards" to write generic rules such as "When a bathroom becomes empty turn off all the lights of the bathroom". A more detailed description of the base language can be found in [see García-Herranz et al. (2008)].

Regarding the base language, a user study has been conducted to measure the adequateness of the triggers–conditions–actions structure and the event–free representation of the world to end–user programming. The study was conducted over 30 Spanish speaker subjects, each of which received by email a short description of a

hypothetical smart home (described through some plans)[1] . Participants were categorized into two groups: those with programming experience (P) and those without it (NP).

They were told that the home understands rules constructed by filling a template with three boxes: triggers (WHEN), conditions (IF) and actions (THEN), as long as they refer to the elements present in the plans, and were given two simple examples. Secondly, they were asked to write rules codifying the automatic behaviors of 5 animations showing scenes recorded in the automated house, e.g. turning off the stove when it has been unused for three minutes or turning on and off the lights as people enter or leave the rooms (see [Figure 1]).

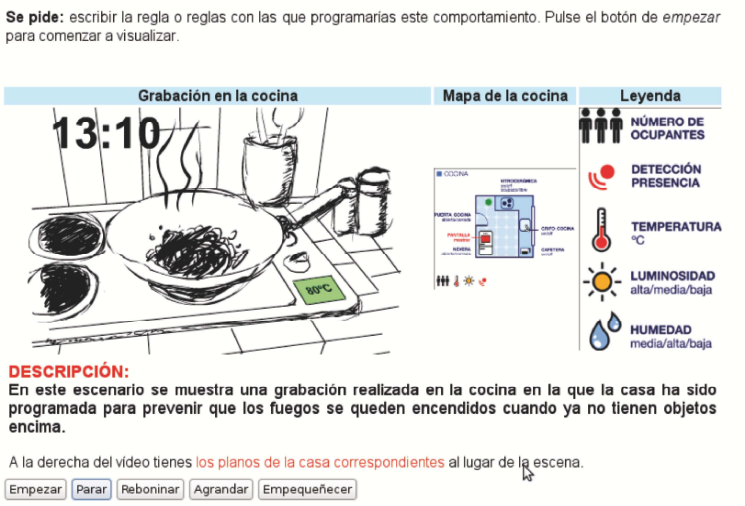

Figure 1: Screenshoot of an exercise of the end–user study. It shows the animation and plans of the elements involved in the animation in the middle of the web page and a short description of what is the video showing below it.

Their answers were evaluated by an expert to measure: (I1) Grammar: they use only elements present in the plans, (I2) Differentiation: they separate triggers and conditions in different sets and (I3) Identification: they correctly assign these sets to their respective boxes. Performance was measured in a three value scale (G for good, M for medium and B for bad). I1 indicates the naturalness of context representation while I2 and I3 measure the adequateness of the triggers–conditions–actions structure. The results are summarized in Table 1.

[1]Questionnaire, plans and videos can be accessed from http://amilab.ii.uam.es:8080/encuesta

	I1			I2			I3		
	G	M	B	G	M	B	G	M	B
NP	63.75%	28.75%	7.50%	87.50%	6.25%	6.25%	57.5%	3.75%	38.75%
P	60.00%	27.69%	12.31%	90.77%	6.15%	3.08%	80.0%	9.23%	10.77%
p		0.17			0.68			0.0029	

Table 1: I1 (Grammar), I2 (Differentiation of Triggers/Conditions) and I3 (Identification of Triggers/Conditions) results (in % for G=Good, M=Medium, B=Bad) for end–users with (P) and without (NP) programming knowledge and their corresponding Mann–Whitney U test p–value.

According to the results obtained, we extracted the following results: a) Even though P performs better than NP in I1, there is no significant statistical difference between both groups (p–value of Mann–Whitney U test of 0.17) obtaining a G 63.75% and 60.00%. b) I2 resulted an easy task for both groups: 87,50% of NP obtained a G (2,81 ± 0,53 in average) as 90,77% of P (2,88 ± 0,41), showing no significant statistical difference between their performance (p–value of 0.68) and c) I3 was complicated for NP: 57,50% obtained a G against the 38,75% with a B (2,19 ± 0,96). It is statistically significant that P perform better in I3 (80,00% of P obtained a G) than NP (p–value of 0.0029).

The great majority of tasks were solved using just one or two rules (both by NP and P), while only one participant used more than three rules to encode a task. Every scenario was encoded using less than three minutes on average.

In addition, users were asked to imagine two more scenarios, defining the elements they need for them and creating their corresponding rules. Since I1 resulted much better in this case than in the previous one we conclude that the bad performance of I1 was due to the unfamiliar home we were presenting, making difficult to remember the names and elements of a complete house within minutes. I2 and I3 present more interesting conclusions. First, that **the differentiation between triggers and conditions can be made naturally both by programmers and non programmers**. Second, that **the Spanish words for "when" and "if" are semantically close (as they are in English) and may lead to misprints among users unfamiliar with the inflexibility of computing languages** e.g. sentences such as "when I enter the house, if it is empty..." can be also expressed as "If I enter the house when it is empty...". Thus, while **triggers and conditions are easily differentiated, natural language does not make their identification easy to non–programmers**.

4.2 Isolating complexity: Time–dependent actions

Some scenarios require more expressiveness than others, but adapting the whole language to deal with it spreads their complexity to the whole language. Thus, we isolated complexity in the elements requiring it, so users only need to deal with it when needed. Additionally, complexity is expressed in the base language terms, making easy to understand and evolve to complex concepts when the basics are known (see [Figure 2]).

As an example of complexity, not every indirect controlled action is to be

executed immediately, some contexts trigger special situations in which the required action depends on some time factor e.g. "Turn off the oven in 20 minutes". Other research has focused on temporal reasoning by embedding time concepts in the whole representation and reasoning system [see Augusto and Nugent (2004)], making it especially suited to deal with time issues but messy to deal with time–independent scenarios.

In our approach, we isolated time constraints into a special kind of action: the *TIMER*. The TIMER is composed of four different parts *ending time*, *on–running rules*, *on–load rules* and *on–finished rules*. The ending time represents the ending time horizon (e.g. "in three minutes", "11:35, July 24th 2008" or "infinity"). The on–running rules is a set of rules (same grammar as the base language) running from the moment the TIMER was started until it is stopped. On–running rules can use the TIMER status as part of the context, being able to restart, pause or stop it according to context or use its value in the conditions. Finally, the on–load and on–finished rules are sets of rules to be executed when the TIMER is initiated/ended, respectively.

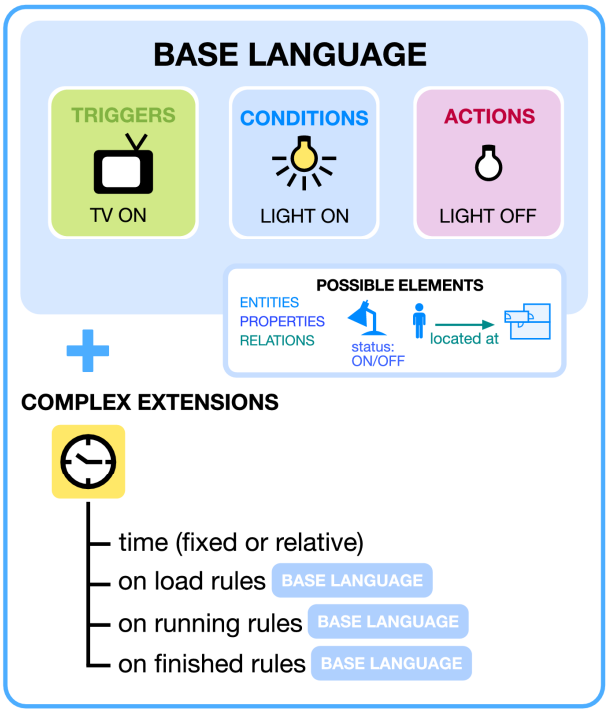

Figure 2: Triggers–Conditions–Actions structure of the base language and description of the TIMER complex extension.

TIMERS can be used in a very complex and powerful way. UIs may hide the low–level details of the *TIMER* to allow end–users to easily use them. This is the case of a GUI we developed to allow supervisors of Alzheimer patients to program alerts and reminders to guide and control their patients activities (e.g. "the pills must be taken at

least 30 minutes before eating").

4.3 A programming example

The base language allows the user to build simple rules, imposing a triggers–conditions–actions structure in which any element, property or relation of the blackboard can be used equally (see [Rule 1]).

Rule 1 Example of a rule that, when the alarm clock sounds, opens the drapes and turns on the radio.

```
device:alarm_clock:status ::
        device:alarm_clock:status = ON
=>
        device:drapes:status := OPEN &&
        device:radio:status := ON
;
```

Generic rules can be created through the use of wildcards. The special symbols * and $ are used to manage and filter sets of entities instead of a single entity in any part of the ECA–rule. They can replace the *entity* or *property* in the pattern *type*:*entity*:*property*. Thus, while *light*:*lamp*_1:*status* refers to "lamp_1" status, *light*:*:*status* refers to every lamp's status or, more accurately, to any lamp status; e.g. the condition *light*:*:*status=ON* would be the set of all the lights turned on (evaluating to false if it is the empty set ∅, and true otherwise). While the event *light*:*:*status* can be translated as *if any light changes its status...*, the symbol * acts as a variable holding the set of matching entities. This set can be accessed through the use of $ followed by the id of the * they have to access. This id is assigned automatically to * from left to right according to the order of appearance in the rule, starting from 0. If a condition has a $ on the left hand side (LHS), the $ set would be filtered to just those elements evaluating the condition to true. Thus, the conditions "*light*:*:*locatedat* = *room*:*lab_b*403&&*light*:$0:*status* = 1" will take all the lights turned on in lab b403 by first obtaining all the lights in b403 and then filtering this set to those turned on. Wildcards can be used by experienced programmers as filters to create more complex rules. An example of a simple rule using wildcards can be seen in Rule 2])

Rule 2 Example of a rule that, when a window is opened, if nobody is in the house it turns on the main alarm. Note that the window must have be opened from outside since nobody is in the house

```
window:*:status ::
        window:$0:status = OPEN &&
space:myhouse:habitants = 0
=>
alarm:main_alarm:status := ON
;
```

On the other hand, the *TIMER* action can be used to create time dependent behaviors (see [Rule 3]). Experienced users (or novice users if the programming UI interfaces abstracts the low level details) can use *TIMERS* to create more complex behaviors, define composite events or different event consumption policies.

Rule 3 Example of a rule codifying *When the egg has been boiling in for 12 minutes, turn off the stove*. This rule takes on account that the egg may be removed or the water stopped boiling before the 12 minute period elapsed, for what the minute counting must be restarted, when the conditions meet again, from the point it was left

```
# TRIGGERS
device:pot:contains || device:pot:boiling ::
# CONDITIONS
device:pot:contains = egg:egg && device:pot:boiling = 1 =>
# ACTION(S)
TIMER 12m 1
# on finished rules
{
device:stove:status := 0 ;
}
# on running rules
{
device:pot:contains :: device:pot:contains != egg:egg
```

```
=> TIMER.pause ;
device:pot:boiling :: device:pot:boiling != 1
=> TIMER.pause ;
device:pot:contains || device:pot:boiling    ::
device:pot:contains = egg:egg &&
 device:pot:boiling = 1 && TIMER.pause
 =>TIMER.start ;
}
 ;
```

The examples of this paper are shown in their kernel code version. Since the kernel language is designed with the end–user in mind, using natural programming structures and isolating complexity, different UIs can be easily created to deal with different interaction scenarios or degrees of expertise. At this moment three different UIs have been implemented: a drag and drop advance interface (see [Figure 3(a)] and [Figure 3(b)]), a simple GUI based on word tokens (see [Figure 3(c)]) and a menu based web interface to easily create reminders for assisted elders[2] (see [Figure 3(d)]).

4.4 Structuring and Managing preferences

Even though rules are the basic programming unit, preferences may be composed by many rules. This is represented in our system by *agents*, a software structure holding and executing a set of rules, irrespective of the UI used to program them. They are a way for users to organize and manage their preferences rather than AI entities. Contrary to other systems, forcing activity or space bundles for the rules (e.g. Youngblood et al. (2005)), we believe that the organization of rules must be up to end–users. Knowing their preferences better than developers do, they may group rules according to many bundles (e.g. the object they affect, an activity, a space, a person or any combination of them). Forcing users to group their behaviors in any fixed way may result in stressful situations. However, some kind of mechanism for organizing and managing preferences is necessary.

[2]Intelligent Reminders can be accessed from http://anahita.ii.uam.es/reminder/

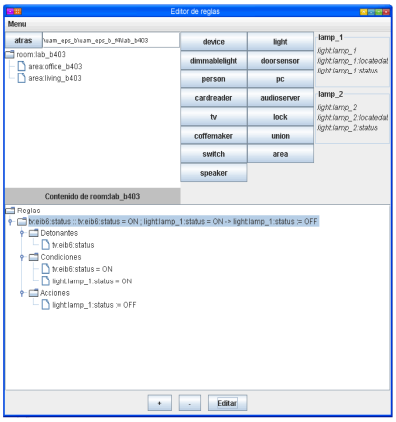

Figure 3(a): Location based navigator window of the GUI rule creation tool. Coded by Carlos Pimentel.

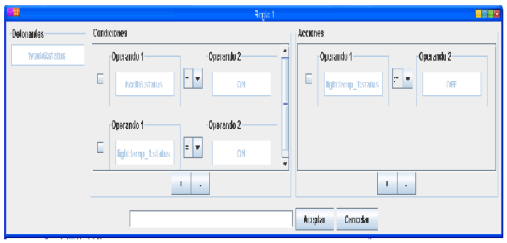

Figure 3(b): *New rule* window of the GUI rule creation tool. Elements are dragged from the navigator window and dropped in the corresponding area of this one (triggers, conditions or actions).

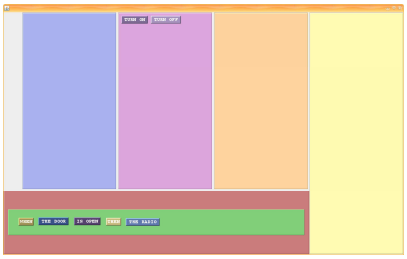

Figure 3(c): Base layout of the Magnet Poetry GUI, showing the different zones for different types of magnets (nouns, verbs, values and links) and the working zone at the bottom. Partially coded by Amanda Vidal.

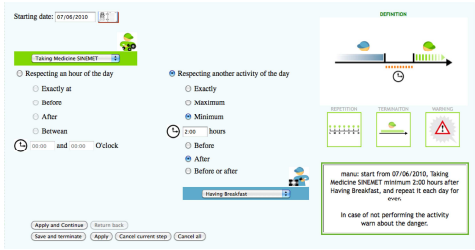

Figure 3(d): Snapshot of the Personal Ambient Intelligent Reminders web application for creating time–based Reminders for assisted living. Created by Dr. Leila Shafti.

Figure 3: Snapshots of different GUIs to create rules.

4.5 Reproducing hierarchies

Multiple users inhabiting the same space makes the interaction dependent on the rest of the users' preferences. Any system for home automation ignoring this matter would fail at some point.

Co–living is a complex human–dependent problem and humans have developed a wide range of hierarchies to deal with it. The structure of the hierarchy (e.g.

centralized, pyramidal or decentralized), the generating principles and the acceptance reasons depend on the particular scenario. Thus, we believe that any system intended to manage hierarchies must take into account both the diversity of structures and the human generation–acceptation factor if it is to succeed. That is, instead of imposing a hierarchy structure **we should provide tools for end–users to build their own**.

The replication of personal hierarchies is of special significance in an environment in which preferences are automated since the same rules that used to coordinate them have to be automated too. Our system takes advantage of the separation between the context layer (i.e. the Blackboard) and the logical layer (i.e. agent–rules system) to deal with different types of prioritization policies (see [Figure 4]). Thus, while the context layer allows preprogrammed device default policies (see [Haya et al. (2006)], [Esquivel et al. (2007)]), the interation or logic layer allows programmable prioritization policies, as indirect control commands. A more detailed description of this filter mechanism can be found in [see García-Herranz et al. (2009)]. As for the scope of this paper we will explore in more detail the use of the agent–rules system as a mechanism for end–users to express their coordination preferences using the same tools they use to express their automation preferences.

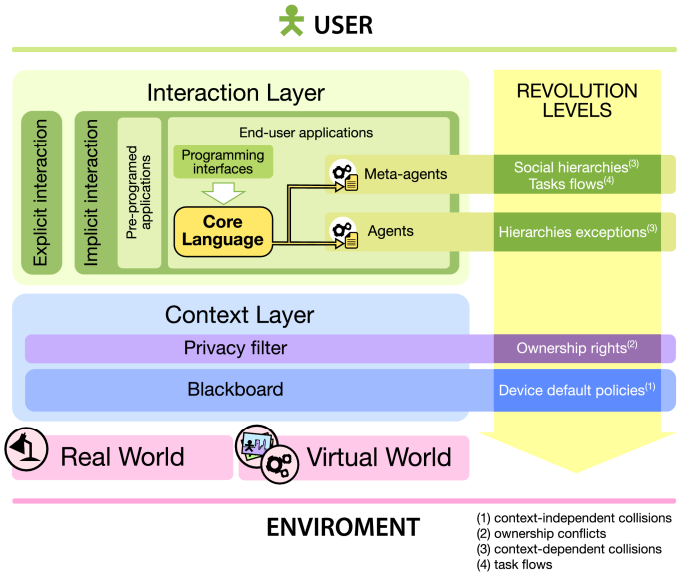

Figure 4: Layers and conflict resolution flow diagram for indirect control. This article focus in the rule–based multi–agent mechanism of the Interaction layer. Agents are software structures which inference process is drive by a set of rules that can be generated from different UI. They also present a natural mechanism to allow users to create coordination rules to solve conflicts between different indirect control commands.

In addition, agents are represented with a set of properties designed to help both organizing and managing the preferences they hold. Those are: *status*, to activate or deactivate the agent, *name*, to easily identify it, *owner*, referring to the person(s) for

which it works and *task*, the abstract goal for which it is programmed. In addition, its *location* (the physical bounds within which it acts) may be represented through a relation. Finally, a relation *affects* is automatically created between the agent and all entities affected by its rules (those in the *actions* part). Thus, users can create rules that activate/deactivate agents of a person, in a particular location, affecting a device, or associated with an scenario, automating their own coordination structures. Therefore, hierarchies are not constrained to be based in social, task or device factors, but they can combine all of them. Their structure can be highly organized or extremely loose, multilayer or singled layered, according to the natural hierarchies of the people involved in the environment. Moreover, the coordination structure is reflected in the Blackboard as the graph created by the persons, their agents and the relations affects (see [Figure 5]).

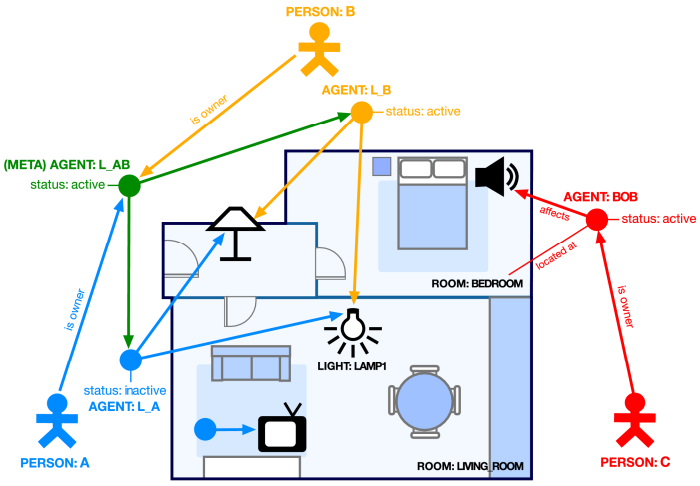

Figure 5: An example of the control graph created by the connections between people and their agents (is_owner), and the agents with the objects they affect (affects).

In this sense, each agent is represented in the blackboard as another entity of the environment, i.e. a virtual assistant. Thus, users can define rules whose actions refer to or affect an agent, as if it was just another light. Agents holding this type of rules are no different from any other agent but can be thought of as meta–agents (see [Rule 4]).

Rule 4 Example of a meta–agent rule activating every inactive agent in a loca- tion when its owner enters that location. Note that, when using wildcards (*), $0 refers to the first * appearing in the rule, $1 to the second and so on.

```
person:*:locatedAt ::
agent:*:locatedAt = person:$0:locatedAt
```

```
 && agent:$1:owner = person:$0
&& agent:$1:status = INACTIVE
=>
agent:$1:status := ACTIVE
 ;
```

Whether a group of persons will encode their preferences together in rules considereing their hierarchies (e.g. "When A enters the house, if B is not present, turn on the TV") or will split them in different agents and coordinate them through meta–agents(e.g. "When B is in the house deactivate A's TV agent") will be up to the group and its idiosyncrasy.

5 Deployment experiences and conclusions

The system is currently running in three laboratories: exploring Ambient Intelligence in personal environments (AmiLab, Universidad Autónoma of Madrid, Spain), teaching environments (ITSZN, Zacatecas, Mexico) and security environments (Indra, Madrid, Spain), allowing several real experiences with users with computer background (with and without programming knowledge). In addition, the system has been tested as a tool to easily combine and seemlessly merge in the environment other state of the art ubicomp technologies.

From these experiences, the most important lessons learned, regarding indirect control in different environments, can be summarized as follows.

First, in personal environments users tend to have a more limited (in scale) and varied idea of their preferences. This resulted in a wide variety of agents, normally dedicated to simple tasks (such as controlling the coffeemaker) within a wide diversity of bundles, added and removed organically, as new preferences arose. This forced our system to **allow an organic evolution of the indirect control structures**, provide some means of **scrutability** and **support a flexible organization of behaviors** to allow users to decide the bundles for their preferences, leaving structure to them.

Second, teaching environments (and working environments in general) presented an example of impersonal spaces, governed more strictly, in which preferences depend less on the individuals than in their roles. Thus, rules hardly ever refer to a particular person or a particular room but they tend to express general preferences such as *"When a teacher enters a room in which a class is taking place..."*. Thus, **we had to provide means to express generality** and to filter generality according to context as e.g. *"...any unused screen of a room participating in the same remote class should show the image of the room in which the teacher is located"*.

Finally, security environments, as an example of specialized spaces, are normally managed by experienced users with strict requirements for more complex preferences. In this sense we found that many scenarios required a **powerful event algebra** for creating composite events, as well as to define **different event consumption policies**. This problem was solved with the *TIMERS*, keeping in mind that the **flexibility and**

expression power must not interfere with the simplicity required to deal with novice end–users. Timers, sequences, periods or probability present great challenges of expression. Isolating their complexity and reproducing already used structures to build them allows to to balance the language's description power with its simplicity.

Along with their diversity, Intelligent Environments are heterogeneous spaces and any system for indirect control must be designed to **integrate and easily incorporate new technologies to the environment**. For doing so, **separating the environment representation and the programming system** has proved extremely useful. This has been experienced in three experiments combining state of the art systems into an already existing Intelligent Environment and using the rule–based agent mechanism to seemlessly control them.

Firstly, to test **the integration of new hardware technologies** we integrated the Phidgets system [see herranz07easingAugmenting] to allow user to easily expand their environments not only with new software behaviors but with new hardware too.

Secondly, to test **the integration of new software technologies**, we incorporated the anthropomorphic virtual character "Maxine" [see seron06maxineppt] of the University of Zaragoza (Spain) to deliver messages in the environment. What to say, when to say it and with which mood were behaviors programmed through the agent mechanism.

Finally, to test **the integration of complex hardware–software systems**, we combined the Smart–its technology and steerable projection system [see molyneaux2007cooperative] of Lancaster University (UK). We tested the integration by creating a cooking scenario in which the system guided the user to cook by projecting over the different elements involved in the recipe (e.g. salt, pans or stove). The logic of the system was delivered by the agents mechanism while the new sensing and intelligent projection capabilities was brought by Lancaster University's work. All merging and programming was done easily and fast while most of the time was spent in side–work such as creating the hardware (pans, stove, salt...).

In summary, even though no application is as efficient as that designed by professionals, we believe that any ubicomp environment should be programmed by their inhabitants in an application–independent way, bringing together every controllable element to the end–user. For doing so, the concept of programming must be kept static to users, no matter the UI they use to program, and carefully balance flexibility and simplicity to deal with the heterogeneity of environments, preferences and skills on Intelligent Environments.

Acknowledgments

This work has been partially funded by the following projects: HADA (Ministerio de Ciencia y Educación de España, TIN2007-64718), Vesta (Ministerio de Industria, Turismo y Comercio de España, TSI-020100-2009-828) y eMadrid (Comunidad de Madrid, S2009/TIC-1650). We thank Dr. David Molyneaux, for many comments and suggestions and Dr. Leila Shafti, Amanda Vidal and Carlos Pimentel for their work with the GUIs.

References

Augusto, J. C., Nugent, C. D.: "The use of temporal reasoning and management of complex events in smart homes"; R. L. de Mántaras, L. Saitta, eds., ECAI; 778–782; IOS Press, 2004.

Cai, Y., Klein-Seetharaman, J.: "Ambient intelligence for scientific discovery"; CHI'04 extended abstracts on Human factors in computing systems; 1706; ACM, 2004.Davidoff, S., Lee, M., Zimmerman, J., Dey, A.: "Socially-aware requirements for a smart home"; Procs of the International Symposium on Intelligent Environments; 41–44; 2006.

Dey, A., Sohn, T., Streng, S., Kodama, J.: "icap: Interactive prototyping of context-aware applications"; Lecture Notes in Computer Science; 3968 (2006), 254–271.

Esquivel, A., Haya, P. A., García-Herranz, M., Alamán, X.: "Managing pervasive environmentprivacyusingthe 'fairtrade' metaphor";InternationalWorkshop on Pervasive Systems, PerSys 2007; 2007.

Francisco Serón, S. B., Cerezo, E.: "Maxineppt: Using 3d virtual characters for natural interaction"; II International Workshop on Ubiquitous Computing and Ambient Intelligence (wUCAmI'2006); Puertollano, Ciudad Real, Spain, 2006.

Gajos, K., Fox, H., Shrobe, H.: "End user empowerment in human centered pervasive computing"; First International Conference on Pervasive Computing, Pervasive 2002; 134–140; 2002.

García-Herranz, M., Haya, P., Alamán, X.: "Easing the smart home: Translating human hierarchies to intelligent environments"; IWANN'2009. Lecture Notes in Computer Science; volume 5517 of LNCS; 1529–1544; Springer-Verlag, Salamanca, Spain, 2009.

García-Herranz, M., Haya, P. A., Alamán, X., Martín, P.: "Easing the smart home: augmenting devices and defining scenarios"; 2nd International Sym- posium on Ubiquitous Computing & Ambient Intelligence - 2007; 2007; best paper award.

García-Herranz, M., Haya, P. A., Esquivel, A., Montoro, G., Alamán, X.: "Easing the smart home: Semi-automatic adaptation in perceptive environments"; Journal of Universal Computer Science; 14 (2008), 9, 1529–1544.

Hague, R.: "End-user programming in multiple languages"; Technical report ucam-cl-tr-651, phd thesis; University of Cambridge, Computer Laboratory (2005).

Haya, P. A., Montoro, G., Alamán, X.: "A prototype of a context-based archi- tecture for intelligent home environments"; International Conference on Cooperative Information Systems (CoopIS 2004); volume 3290 of Lecture Notes in Computer Science (LNCS); Larnaca, Cyprus, 2004.

Haya, P. A., Montoro, G., Esquivel, A., García-Herranz, M., Alamán, X.: "A mechanism for solving conflicts in ambient intelligent environments"; Journal Of Universal Computer Science; 12 (2006), 3, 284–296.

Intille, S. S., Larson, K.: "Designing and evaluating supportive technology";

IEEE/ASME International Conference on Advanced Intelligent Mechatron- ics 2003; 501–518; IEEE Press, 2003.

Kulkarni, A.: A reactive behavioral system for the intelligent room; Ph.D. thesis; Massachusetts Institute of Technolohy (2002).

Molyneaux, D., Gellersen, H., Kortuem, G., Schiele, B.: "Cooperative augmen- tation of smart objects with projector-camera systems"; J. Krumm, G. D. Abowd, A. Seneviratne, T. Strang, eds., UbiComp; volume 4717 of Lecture Notes in Computer Science; 501–518; Springer, 2007.

Mozer, M. M.: "The neural network house: An environment that adapts to its inhabitants"; Proceedings of the AAAI Spring Symposium on Intelligent Environments; AAAI Press, 1998.

Myers, B. A., Pane, J. F., Ko, A.: "Natural programming languages and environments"; Commun. ACM; 47 (2004), 9, 47–52.

Papert, S.: Mindstorms: Children, Computers, and Powerful Ideas; Basic Books, New York, 1980.

Repenning, A., Ioannidou, A.: "What makes End-User Tick? 13 Design Guidelines"; chapter 4, 51–85; Human-Computer Interaction Series, Vol. 9; Springer- Verlag, 2006.

Rodden, T., Crabtree, A., Hemmings, T., Koleva, B., Humble, J., Akesson, K.- P., Hansson, P.: "Between the dazzle of a new building and its eventual corpse: assembling the ubiquitous home"; D. Benyon, P. Moody, D. Gruen, I. McAra-McWilliam, eds., Conference on Designing Interactive Systems; 71–80; ACM, 2004.

Román, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R., Nahrstedt, K.: "A middleware infrastructure for active spaces"; IEEE Pervasive Computing; 1 (2002), 4, 74–83.

Schmidt, A.: "Implicit human computer interaction through context"; Personal and Ubiquitous Computing; 4 (2000), 2/3.

Truong, K. N., Huang, E. M., Abowd, G. D.: "Camp: A magnetic poetry interface for end-user programming of capture applications for the home"; N. Davies, E. D. Mynatt, I. Siio, eds., Ubicomp; volume 3205 of Lecture Notes in Computer Science; 143–160; Springer, 2004.

Youngblood, G. M., Cook, D. J., Holder, L. B.: "Managing adaptive versatile environments"; Pervasive and Mobile Computing; 1 (2005), 4, 373–403.

Zhang, T., Brügge, B.: "Empowering the user to build smart home applications"; Toward A Human-Friendly Assistive Environment: ICOST'2004, 2nd International Conference on Smart Home and Health Telematics; 170; IOS Press, 2004.