Creating an Ambient-Intelligence Environment Using Embedded Agents

Hani Hagras, Victor Callaghan, Martin Colley, Graham Clarke, Anthony Pounds-Cornish, and Hakan Duman, *University of Essex*

The Essex intelligent dormitory, iDorm, uses embedded agents to create an ambient-intelligence environment. In a five-and-a-half-day experiment, a user occupied the iDorm, testing its ability to learn user behavior and adapt to user needs. The embedded agent discreetly controls the iDorm according to user preferences.

Ambient intelligence is an exciting new information technology paradigm in which people are empowered through a digital environment that is aware of their presence and context and is sensitive, adaptive, and responsive to their needs.[1] Ambient-intelligence environments are characterized by their ubiquity, transparency, and intelligence. In these environments, a multitude of interconnected, invisible embedded systems, seamlessly integrated into the background, surround the user. The system recognizes the people that live in it and programs itself to meet their needs by learning from their behavior.[1]

To realize the ambient-intelligence vision, people must be able to seamlessly and unobtrusively use and configure the computer-based artifacts and systems in their ubiquitous-computing environments without being cognitively overloaded.[1] The user shouldn't have to program each device or connect them together to achieve the required functionality. The complexity associated with the number, varieties, and uses of computer-based artifacts requires that we design a system that lets intelligence disappear into the infrastructure of active spaces (such as buildings, shopping malls, theaters, and homes),[2] automatically learning to carry out everyday tasks based on the users' habitual behavior.

Our work focuses on developing learning and adaptation techniques for embedded agents. We seek to provide online, lifelong, personalized learning of anticipatory adaptive control to realize the ambient-intelligence vision in ubiquitous-computing environments. We developed the Essex intelligent dormitory, or iDorm, as a test bed for this work and an exemplar of this approach.

Intelligent embedded agents

Embedded intelligence refers to including some capacity for reasoning, planning, and learning in an artifact. Embedded computers that contain such an intelligent capability are normally referred to as *embedded agents*[2] and are intrinsic parts of *intelligent artifacts*. These autonomous entities typically have a network connection, thereby facilitating communication and cooperation with other embedded agents to form multi-embedded-agent systems.

Embedded agents in the form of mobile robotic agents can learn and adapt their navigation behaviors online.[3] However, we concentrate on embedded agents in ubiquitous-computing environments that will help us realize the ambient-intelligence vision. Each embedded agent is connected to sensors and effectors, comprising a ubiquitous-computing environment. The agent uses our fuzzy-logic-based *Incremental Synchronous Learning* (ISL) system to learn and predict the user's needs, adjusting the agent controller automatically, nonintrusively, and invisibly on the basis of a wide set of parameters (which is one requirement for ambient intelligence).[4] Thus, we need to modify effectors for environmental variables (such as heat and light) on the basis of a complex, multidimensional input vector. An added control difficulty is that people are essentially nondeterministic and highly individual. Because the embedded agents are located on small embedded computers with limited processor and memory abilities, any learning and adaptation system must deal with these computational limitations.

Most automation systems, which involve minimal intelligence, use mechanisms that generalize actions across a population—for example, setting temperature or loudness to the average of many peoples' needs.

1541-1672/04/$20.00 © 2004 IEEE
Published by the IEEE Computer Society

Related Work

A growing number of research projects are concerned with applying intelligent agents to intelligent inhabited environments and intelligent buildings. In Sweden, Paul Davidsson and Magnus Boman used multiagent principles to control building services.[1] These agents are based on the artificial intelligence thread that decomposes systems by function rather than behavior. Their work does not address issues such as occupant-based learning. In Colorado, Michael Mozer uses a soft-computing approach—neural networks—focusing solely on the intelligent control of lighting within a building.[2] Mozer's system, implemented in a building with a real occupant, achieved a significant energy reduction, although this was sometimes at the expense of the occupant's comfort. Work at the Massachusetts Institute of Technology on the HAL project concentrated on making the room responsive to the occupant by adding intelligent sensors to the user interface.[3] Context-aware systems such as the Aware Home[4] at the Georgia Institute of Technology represent a large body of current research effort but differ from our work in that they are more concerned with time-independent context rather than temporal history or learning, which are central issues in our work.

Other high-profile intelligent-environment projects also exist, such as the Microsoft Smart House, BT's Tele-care, and Cisco's Internet Home.[5] However, most of these industrial projects, including home automation technologies such as Lonworks and X10, are geared toward using networks and remote access with some smart control (mostly simple automation), with sparse use of AI and little emphasis on learning and adaptation to the user's behavior.

References

1. P. Davidsson and M. Boman, "Saving Energy and Providing Value-Added Services in Intelligent Buildings: A MAS Approach," *Proc. 2nd Int'l Symp. Agent Systems and Applications and 4th Int'l Symp. Mobile Agents* (ASA/MA 2000), Springer-Verlag, 2000, pp. 166–177.
2. M. Mozer, "The Neural Network House: An Environment That Adapts to Its Inhabitants," *Proc. Am. Assoc. Artificial Intelligence Spring Symp. Intelligent Environments*, AAAI Press, 1998, pp. 110–114.
3. M. Coen, "Design Principles for Intelligent Environments," *Proc. 15th Nat'l Conf. Artificial Intelligence* (AAAI 98), AAAI Press, 1998, pp. 547–554.
4. G. Abowd et al., "Context-Aware Computing," *IEEE Pervasive Computing*, vol. 1, no. 3, July–Sept. 2002, pp. 22–23.
5. A. Sherwin, "Internet Home Offers a Life of Virtual Luxury," *The Times*, 3 Nov. 1999, p. 10.

However, to achieve the ambient-intelligence vision, any type of intelligence applied to personal artifacts and spaces must be particular to the individual.[2] Furthermore, any agent serving a person must always and immediately carry out any requested action—that is, to achieve the responsive property implied in the ambient-intelligence vision, people must always be in control, subject to overriding safety considerations.[1] The embedded-agent learning technique we've adopted can particularize its actions to individuals and immediately execute user commands. We are testing our embedded agent in the iDorm.

Intelligent inhabited environments and intelligent buildings

Intelligent inhabited environments are spaces such as cars, shopping malls, homes, and even our bodies that respond "thoughtfully" to our needs. Such environments would consist of a multitude of possibly disconnected active spaces providing ubiquitous access to system resources according to the user's current situation. Such environments promise a future where computation will be freely available everywhere, similar to the availability of batteries and power sockets today. These intelligent environments will personalize themselves in response to our presence and behavior.

Intelligent buildings are precursors to such environments.[2] A typical container environment for ubiquitous computing is an intelligent building, possibly a house or office. The heterogeneity, dynamism, and context-awareness in a building make it a good choice to explore ubiquitous-systems design challenges. We view intelligent buildings as computer-based systems, gathering information from various sensors (and other computers) and using intelligent embedded agents to determine various devices' appropriate control actions.[2,3,5] In controlling such systems, we are faced with the imprecision of sensors, the large number of information sources, the lack of adequate models of many of the processes, and the nondeterministic aspects of human behavior. Embedded agents must be able to continuously learn and adapt to the needs of individuals in an intelligent building, while always providing a safe and timely response to any situation.[5] (See the "Related Work" sidebar for other research in this area.)

iDorm

The iDorm (see Figure 1a) is a test bed for ubiquitous-computing environments. We are using the iDorm to test the intelligent-learning and adaptation mechanisms our embedded agent needs with the hopes of realizing the ambient-intelligence vision in ubiquitous-computing environments. As an intelligent dormitory, the iDorm is a multiuse space—that is, it contains areas for varied activities such as sleeping, working, and entertaining—that compares in function to other living or work spaces such as a one-room apartment, hotel room, or office. The iDorm contains the normal mix of furniture found in a study or bedroom, letting the user live comfortably. The furniture (most of which we fitted with embedded sensors) includes a bed, work desk, bedside cabinet, wardrobe, and PC-based work and multimedia entertainment system. The PC contains most office-type programs to support work as well as audio and video services for entertainment (to play music CDs, listen to radio stations using Dolby 5.1 surround sound, and watch television and DVDs).

To make the iDorm as responsive as possible to its occupant's needs, we fitted it with an array of embedded sensors (such as temperature, occupancy, humidity, and light-level sensors) and effectors (such as door actuators, heaters, and blinds).[6] Among these interfaces, we produced the virtual reality system in Fig-

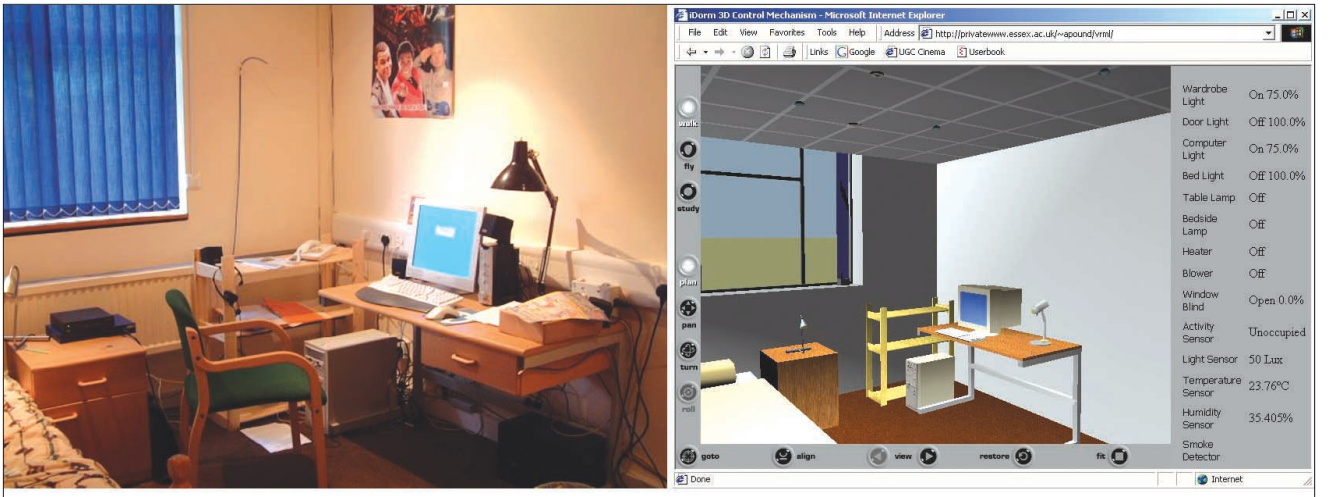


Figure 1. (a) The Essex intelligent dormitory iDorm and (b) the iDorm VRML (Virtual Reality Modeling Language) interface.

ure 1b, which marries the Virtual Reality Modeling Language (VRML) with a Java interface controlling the iDorm. It provides the user with a visualization tool showing the iDorm's current state and enables direct control of the various effectors in the room.

Although the iDorm looks like any other room, the ceiling and walls hide numerous networked embedded devices residing on three different networks: Lonworks, 1-Wire, and IP. They provide the diverse infrastructure present in ubiquitous-computing environments and let us develop network-independent solutions.[6] Because we need to manage access to the devices, gateways between the different networks are critical components in such systems, combining appropriate granularity with security.

Lonworks, Echelon's proprietary network, includes a protocol for automating buildings. Many commercially available sensors and actuators exist for this system. The physical network installed in the iDorm is the Lonworks TP/FP10 network, and Echelon's iLON 1000 Web server provides the gateway to the IP network. This server lets us read and alter the states and values of sensors and actuators via a standard Web browser using HTML forms. Most of the sensors and effectors in the iDorm are connected via a Lonworks network.

The 1-Wire network, developed by Dallas Semiconductor, was designed to connect simple devices over short distances. It offers a range of commercial devices including small temperature sensors, weather stations, ID buttons, and switches. The 1-Wire network is connected to a Tiny Internet Interface board (www.ibutton.com/TINI), which runs an embedded Web server serving the status of the networked devices using a Java servlet. The servlet collects data from the network devices and responds to HTTP requests.

The IP network forms a backbone to interconnect all the networks and other devices, such as the multimedia PC. This PC is the focus for work and entertainment in the iDorm; it also uses the HTTP protocol to display its information as a Web page.

The iDorm's gateway server is a practical implementation of an HTTP server acting as a gateway to each of the room's subnetworks. This shows that by using a hierarchy of gateways, it would be possible to create a scalable architecture across such heterogeneous networks in intelligent inhabited environments and ubiquitous-computing environments.[6] The iDorm gateway server allows a standard interface to all the room's subnetworks by exchanging XML-formatted queries with all the principal computing components. This overcomes many of the practical problems of mixing networks. This gateway server lets the system operate over any standard network such as EIBus or Bluetooth. We could readily develop it to include plug-and-play, letting the system automatically discover and configure devices using intelligent mechanisms.[6] In addition, such a gateway is clearly an ideal point to implement security and data mining associated with the subnetwork. Figure 2 shows the logical network infrastructure in the iDorm.

iDorm's embedded computational artifacts

The iDorm has three types of embedded computational artifacts connected to the network infrastructure. Some of these devices contain agents.

The first type is a physically static computational artifact closely associated with the building. In our case, this artifact contains an agent and thus is termed the *iDorm embedded agent*. This agent receives sensor values through the network, contains the user's learned behavior, and computes the appropriate control actions using the fuzzy ISL system. It then sends them to iDorm effectors across the network. The agent shown in Figure 3a is based on a 68000 Motorola processor with 4 Mbytes of RAM, has an Ethernet network connection, and runs the VxWorks Real Time Operating System.

The agent accesses 11 environmental parameters, some on multifunction appliances:

- Time of day, measured by a clock connected to the 1-Wire network
- Inside room light level, measured by an indoor light sensor connected to the Lonworks network
- Outdoor lighting level, measured by an external weather station connected to the 1-Wire network
- Inside room temperature, measured by sensors connected to the Lonworks and 1-Wire networks
- Outdoor temperature, measured by an external weather station connected to the 1-Wire network

- Whether the user is running the computer's audio entertainment system, sensed by custom code that publishes the activity on the IP network
- Whether the user is lying or sitting on the bed, measured by pressure pads connected to the 1-Wire network
- Whether the user is sitting on the desk chair, measured by a pressure pad connected via a low-power wireless connection to the 1-Wire network
- Whether the window is opened or closed, measured by a reed switch connected to the 1-Wire network
- Whether the user is working, sensed by custom code that publishes the activity on the IP network
- Whether the user is using video entertainment on the computer (either a TV program via WinTV or a DVD using the Winamp program), sensed by custom code that publishes the activity on the IP network

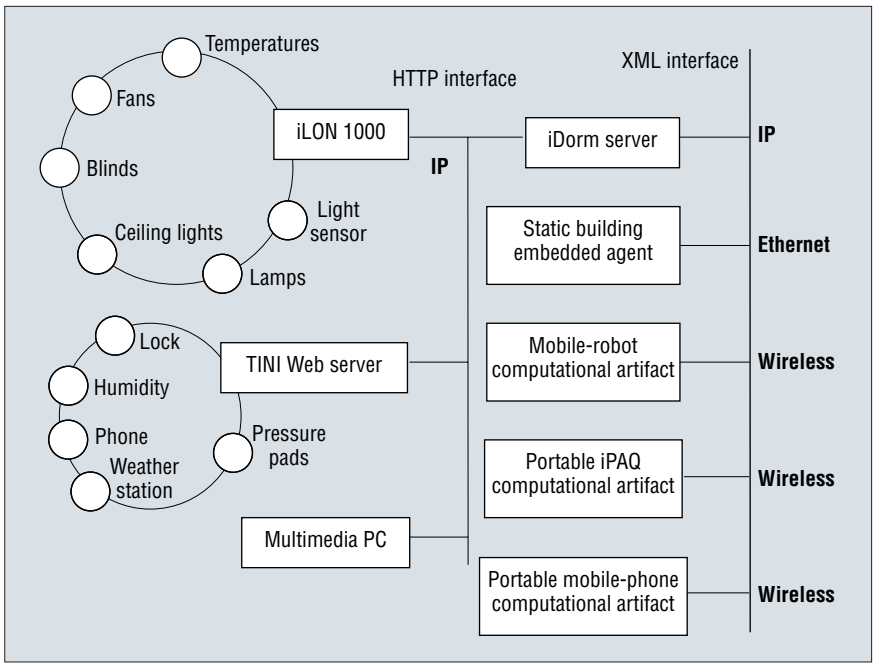


Figure 2. The logical network infrastructure in the iDorm.

The agent controls nine effectors, which are attached to the Lonworks network:

- A fan heater
- A fan cooler
- A dimmable spotlight above the door
- A dimmable spotlight above the wardrobe
- A dimmable spotlight above the computer
- A dimmable spotlight above the bed
- A desk lamp
- A bedside lamp
- Automatic blind status (open or closed, or at an angle)

Other sensors in the room include a smoke detector, a humidity sensor, activity sensors, and a telephone sensor (to sense whether the phone is on or off the hook) as well as a camera to monitor what happens in the iDorm. It's possible to follow (and control) activities in the iDorm via a live video link over the Internet.

The second type of embedded computational artifact is a robotic agent, a physically mobile service robot containing an agent. The robotic agent can learn and adapt robot navigation behaviors online[3] (which is different from the iDorm embedded agent, which seeks to realize ambient intelligence). Figure 3b shows the robot prototype we use in the iDorm. The robot is a servant-gadget for delivering various objects of interest to the iDorm user such as food, drink, and medicine. It has a rich set of sensors (nine ultrasound sensors, two bumpers, and an IR beacon receiver) and actuators (wheels). It uses 68040 Motorola processors and runs the VxWorks Real-Time Operating System.

The robot is equipped with essential behaviors for navigation, such as obstacle avoidance, goal seeking, and edge following. We combined and coordinated these behaviors with a fuzzy coordination module so that the robot could reach a desired location and avoid obstacles. The static embedded agent that controls the iDorm passes and processes the robot's location as an additional input. In

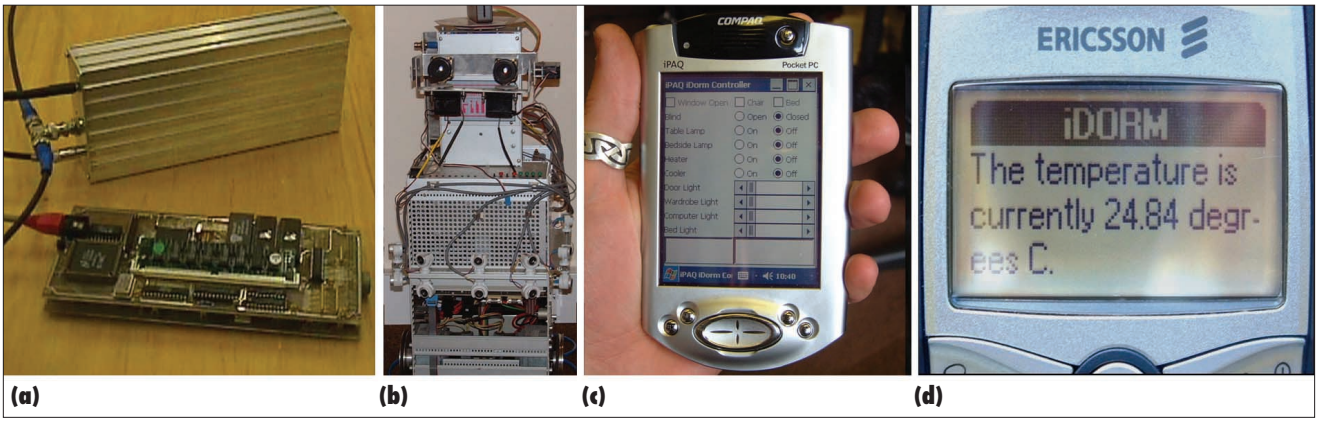


Figure 3. The iDorm embedded computational artifacts include (a) the static iDorm embedded agent, (b) a mobile service robot, (c) a portable iPAQ (pocket PC) interface, and (d) a portable mobile-phone interface.

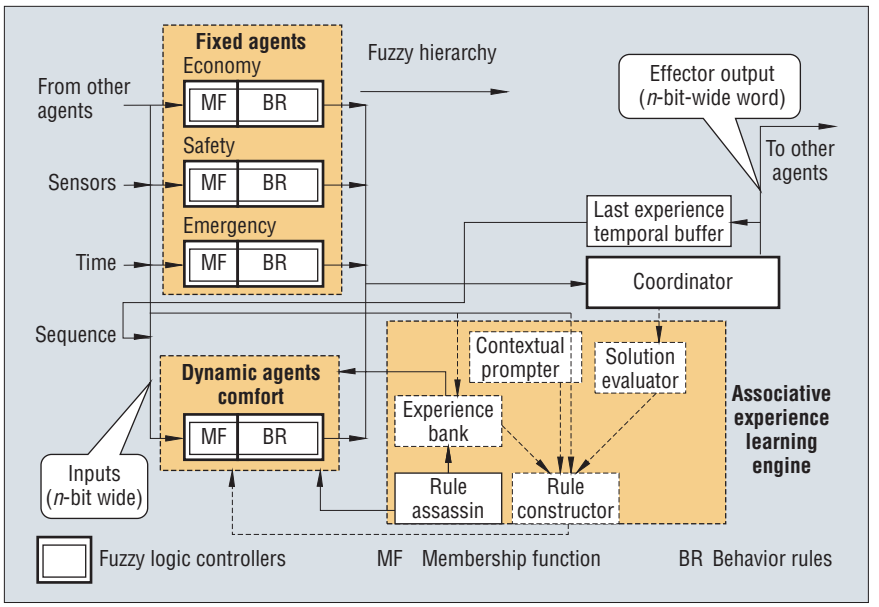


Figure 4. The Incremental Synchronous Learning architecture.

the experimental setup, we use a simplified system in which the robot can go to two locations identified by infrared beacons to pick up objects. After picking up an object, the robot can deliver it to the user and then go to its charging station, which another infrared beacon identifies. The robotic agent sends information about its location to the iDorm agent, and it takes destination instructions from that agent depending on the user's learned behavior. For example, the robot might learn to fetch a newspaper from a specific location whenever it is delivered in the morning.

We implemented the communication between the static embedded agent and the mobile robotic agent via a wireless link. We establish communication by initiating a request from the embedded agent to the mobile agent server. Once the request has been sent, the server passes it to the robotic agent to carry out the task and informs the iDorm embedded agent of the robot's current status. If the task is in progress or not completely finished, the server sends a message indicating that the job is incomplete. Every time the iDorm embedded agent wants to send out a new request, it waits until the robot successfully completes the previously requested job.

The third type of embedded computational artifact is a physically portable computational device. Typically, these are wearable technologies that can monitor and control the iDorm wirelessly. The handheld iPAQ in Figure 3c contains a standard Java process that can access and control the iDorm directly. This forms a type of remote control interface that would be particularly suitable to elderly and disabled users. Because the iPAQ supports Bluetooth wireless networking, it's possible to adjust the environment from anywhere inside and nearby outside the room. It's also possible to interact with the iDorm through mobile phones because the iDorm central server can also support the Wireless Markup Language. Figure 3d shows the mobile-phone wireless application protocol interface, which is a simple extension of the Web interface. Such portable devices can contain agents, but this remains one of our longer-term goals.

We designed the learning mechanism in the embedded agent to learn behaviors relating to different individuals. To achieve this, the embedded agent must be able to distinguish between users in the environment. This is achieved by using an active key button, designed and built by our research team and based on Dallas Semiconductor's 1-Wire protocol. Each user is given an electronic key about the size of a penny. This is mounted onto a key fob and contains a unique identification number inside its 2-Kbyte memory. The user's unique ID number is passed to the iDorm embedded agent so that it can retrieve and update previous rules it learned about that user.

Fuzzy, incremental, synchronous learning technique

In our work, learning is achieved through interaction with the actual environment. We call this *online learning* because adaptive behaviors can't be considered a product of an agent in isolation from the world but can only emerge from a strong coupling of the agent and its environment.[5]

Figure 4 shows the ISL architecture, which forms the learning engine in the iDorm embedded agent. The ISL system aims to provide lifelong learning and adapts by adding, modifying, or deleting rules. It is memory based in that the system can use its previous experiences (held as rules) to narrow down the search space and speed up learning. The embedded agent is an augmented-behavior-based architecture, which uses a set of parallel fuzzy logic controllers (FLCs), each forming a behavior. We use the FLC approach because it's useful when the processes are too complex for analysis by conventional quantitative techniques or when the available sources of information are interpreted qualitatively, imprecisely, or uncertainly[3,5] (this is the case with embedded agents operating in intelligent inhabited environments and ubiquitous-computing environments). For embedded agents, the number of inputs and outputs are usually large, and the desired control behaviors are complex. However, by using a hierarchical assembly of fuzzy controllers, we significantly reduce the number of rules required.[3,5]

In general, we divide the behaviors available to the iDorm embedded agent into fixed and dynamic sets, where the dynamic behaviors are learned from the person's behavior and the fixed behaviors are preprogrammed. We predefined these latter behaviors because they can't easily be learned—for example, the temperature at which water pipes freeze. The fixed behaviors include safety, emergency, and economy behaviors. A safety behavior ensures that the environmental conditions are always at a safe level. An emergency behavior (in case of a fire alarm or another emergency) might open the emergency doors and switch off the main heating and illumination systems. Economy behaviors ensure that energy isn't wasted so that if a room is unoccupied, the heating and illumination will be switched to a sensible minimum value. All these behaviors are fixed but adjustable.

Each dynamic FLC (the comfort behavior in the iDorm case) has one parameter (which is the rule base for each behavior) that we can modify. Also, at the high level, the coordination parameters can be learned.[3,5] Each behavior uses a FLC using a singleton fuzzifier, triangular membership functions, prod-

uct inference, max-product composition, and height defuzzification. We chose these techniques because of their computational simplicity and real-time considerations. The equation that maps the system input to output is

$$Y_t = \frac{\sum_{p=1}^{M} y_p \prod_{i=1}^{G} \alpha_{Aip}}{\sum_{p=1}^{M} \prod_{i=1}^{G} \alpha_{Aip}}$$

where M is the total number of rules, y_p is the point having maximum membership in the pth rule output fuzzy set, $\Pi\alpha_{Aip}$ is the product of the membership functions for each rule's inputs, and G is the number of inputs.

We use a higher-level coordinator FLC to combine the preferences of different behaviors into a collective preference (giving a two-level behavior hierarchy). (Previous work gives additional information about the fuzzy hierarchical architecture.[3,5])

The ISL works as follows: when new users enter the room, they are identified by the active key button, and the ISL enters an initial monitoring mode where it learns the users' preferences during a nonintrusive cycle. In the experimental setup, we used a 30-minute period, but in reality, this is linked to how quickly and completely we want the initial rule base. For example, in a care home, we might want this rule base to be as complete as possible, and in a hotel, we might want this initialization period to be short to allow fast learning. The rules and preferences learned during the monitoring mode form the basis of the user rules, which are reactivated whenever the user reenters the room. During this initialization period, the system monitors the inputs and the user's action and tries to infer rules from the user's behavior. The user will usually act when a set of environmental conditions (an input vector) is unsatisfactory by altering the output vector (for example, the user needs to turn a light on or adjust the heating). Learning is based on negative reinforcement because users will usually request a change to the environment when they are dissatisfied with it.

After the monitoring period, the ISL enters a control mode in which it uses the rules learned during the monitoring mode to guide its control of the room's effectors. Whenever the user behavior changes, it might need to modify, add, or delete some of the rules in the rule base. Thus, the ISL goes back to the nonintrusive cycle to infer rule base changes—that is, to determine the user's preferences in relation to the specific components of the rules that have failed. The user is essentially unaware of this short cycle; such modifications and adaptations are distributed throughout the lifetime of the environment's use, thus forming a lifelong-learning phase.

As in the case of classifier systems, to preserve system performance, we let the learning mechanism replace a subset of the classifiers (the rules in this case). The worst m classifiers are replaced by m new classifiers.[4] In our case, we change all the consequents of the rules whose consequents were unsatisfactory to the user. We find these rules by finding all the rules firing at this situation whose firing strength is $\Pi\alpha_{Aip} > 0$. We replace these rule consequents by the fuzzy set that has the highest membership of the output membership function. We make this replacement to achieve nonintrusive learning, avoiding direct interaction with the user. The set of learned-consequent fuzzy rules is guided by the contextual prompter, which uses sensory input to guide the learning.

This marks a significant difference in our method of classifying or managing rules compared to other work: rather than seeking to extract generalized rules, we try to define particularized ones.

During the nonintrusive monitoring and the lifelong-learning phases, the agent encounters many different situations as both the environment and the user's behavior change. For example, the agent will try to discover the rules needed in each situation guided by the occupant's behavior in response to different temperature and lighting levels inside and outside the room. The learning system consists of different learning episodes; in each situation, the agent will fire only a small number of rules. The model the agent must learn is small, as is the search space. The accent on local models implies the possibility of learning by focusing at each step on only a small part of the search space, thus reducing interaction among partial solutions. The interaction among local models, due to the intersection of neighboring fuzzy sets, means local learning reflects on global performance.[4] Thus, we can have global results from the combination of local models and smooth transition between close models. By doing this, we don't need to learn the complete rule base all at once, only the rules the user needs during the different episodes. This marks a significant difference in our method of classifying or managing rules compared to other work: rather than seeking to extract generalized rules, we try to define particularized ones.

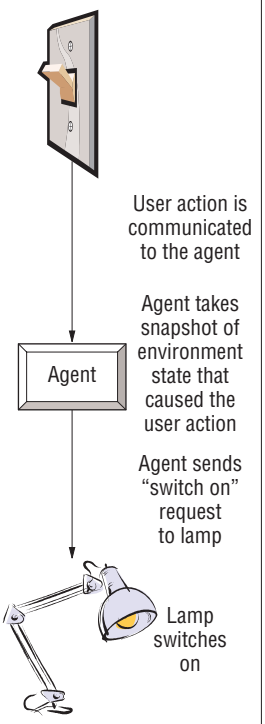
The system has an Experience Bank that stores all the previous occupiers' rule bases. After the initial monitoring phase, the system tries to match the user-derived rules to similar rules stored in the Experience Bank that were learned from other occupiers. The system chooses the rule base that's most similar to the user-monitored actions. By doing this, the system is trying to predict the rules that weren't fired in the initialization session, thus minimizing the learning time as the search starts from the closest rule base rather than starting at random. This action should be satisfactory for the user as the system starts from a similar rule base and then fine-tunes the rules.

Subsequently, the agent operates with rules learned during the monitoring session plus rules that deal with situations uncovered during the monitoring process, which are ported from the most similar user's rule base. All the rules that are constructed and added to the system are symbolized by the Rule Constructor block in Figure 4. The system then operates in the control mode with this rule base until the occupant's behavior indicates that his or her needs have altered; this change is flagged by the Solution Evaluator (that is, the agent is event-driven). The system can then add, modify, or delete rules to satisfy the occupant by briefly reentering the monitoring mode. In this case again, the system finds the rules fired and changes their consequent to the user's action. In this way, the system implements a lifelong-learning strategy.

Because we're dealing with embedded agents with limited computational and memory capabilities, it's difficult to deal with a large number of rules in the rule base. For example, for the iDorm comfort behaviors in our current implementation, a complete rule base contains 62,208 rules. This would lead

(a)

(b)



(c)

Figure 5. The user is (a) using the iDorm's active lock and (b) sleeping in bed. (c) The agent communication path.

to large memory and processor requirements that are unrealistic in embedded agents. So, we limited the number of stored rules to 450 (in our case, the maximum number the agent can store on the onboard memory without exceeding the memory limit or degrading real-time performance). Each rule will have a measure of importance according to how frequently it's used. In calculating this degree of importance, we also include a most-recent-use measure. When the system reaches the memory limit, the Rule Assassin retains rules according to the priority of highest frequency of use, followed by most recently used. If two rules share the same degree of relative rule frequency recall, the system breaks the tie by eliminating the least recently used rule. This action lets the onboard memory store only the most efficient and frequently used rules and reduces degradation of the embedded agent's real-time performance. However, we can store the rules chosen for replacement with the other rules representing the user behavior in an external hard disk, so that the agent can recall them when needed.

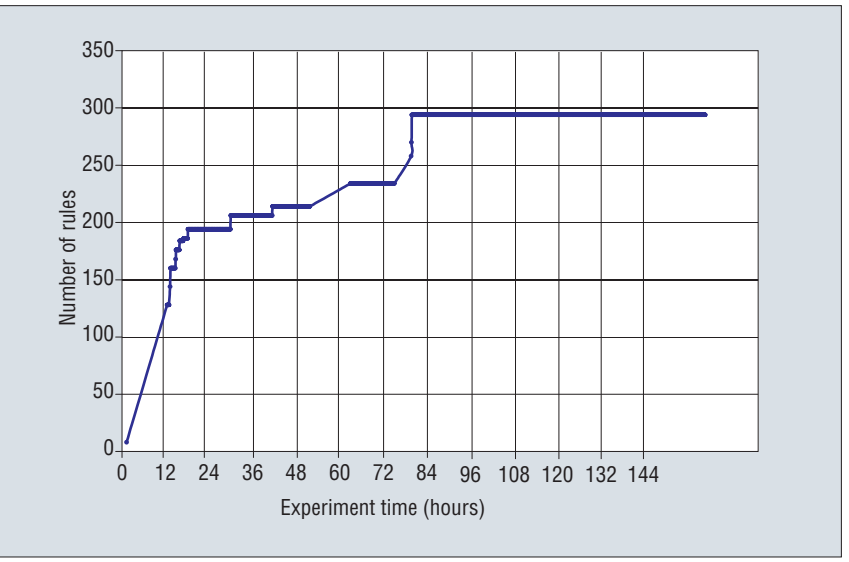


Figure 6. The rules learned plotted against experiment time.

Experimental results

In our experiment, a user occupied the iDorm for five and a half days (132 hours). The system identified the user by his active key button, which operated the active lock (see Figure 5a). In our experiment, the user (shown sleeping in Figure 5b) occupied the iDorm for five and a half days (132 hours). He used the wireless iPAQ to monitor and control the iDorm environment whenever he was dissatisfied with the environment's current state. We recorded a history of the user decisions in a journal. One of our axioms is that "the user is king," by which we mean that an agent always executes the user's instruction immediately, to achieve the responsive property implied in the ambient-intelligence vision, unless safety is compromised. Figure 5c shows this; whenever changes to controls occurred, the iDorm embedded agent received the request, generated a new rule or adjusted a previously learned rule, and allowed the action. We wrote a small parsing tool to convert the text file containing the fuzzy-rule sets into a human-readable format.

At the end of the experiment, we examined the rules in two ways. First, we compared the human-readable rules with the user's journal entries to ensure that the agent had successfully learned the behaviors the user was intending. Second, we compared the number of rules learned over time. We measured the embedded agent's success by monitoring how well it matched the environment to the user's demands. If it did this well, the user intervened less, which resulted in less rule generation over time. If it did this poorly, the user intervened more, which resulted in more rule generation over time. A logging program took a reading of the

number of rules learned by the ISL every five minutes for the experiment's duration; it stored each count along with a time stamp.

Figure 6 summarizes the experiment's results. Several sections of the graph are worth noting. In the first 24 hours, starting with an initial rule base of zero, the system learned many rules in a comparatively short period of time. In fact, in the experiment's first nine hours, the agent learned 128 rules—nearly half the total number of rules learned by the end of the experiment. These results are consistent with the agent learning and making incorrect decisions for the user in the initial stage. However, at the end of the first 24 hours, the agent's learning rate (rules/time) dropped drastically. The agent's reactions required less correction by the user because it made useful decisions about the environment based on the user's requirements. This trend of fewer rules learned over time is consistent across the whole experiment. Hence, the level of comfort the user experienced (in relation to the environment state) was high enough for him not to make an environmental change and consequently alter the learning rate.

The second section of the graph (from 60 to 72 hours) shows a sharp increase in the agent's learning rate. This is explained by the user introducing novel activity into his repertoire of behaviors. It shows that our system, which operates in a lifelong learning mode, adapts to user needs.

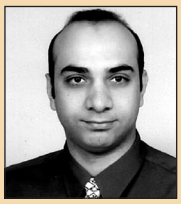
The third section (from 72 to 132 hours) shows that in the experiment's last two days, the agent didn't generate any new rules. The user didn't intervene with the system because he was generally satisfied with the agent's actions.

The agent learned the 280 rules needed to capture this user's behavior over the 132-hour experiment, which demonstrates that our system can learn effectively using the ISL, and it doesn't need to learn the complete rule base (potentially 62,208 rules in the case of the iDorm). Also, over the experimental period, the agent made a significant reduction in the user's need to intervene. Figure 6 shows that the agent had to learn fewer new rules about the user as the experiment progressed. Because this was one of our criteria for measuring the agent's success, the evidence of the continual reduction in the learning rate leads us to conclude that the agent managed to pick out the user's pertinent behavior over time.

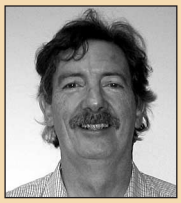
In our previous work, we experimented with different room users.[3] We found that the role of the Experience Bank was important in reducing the time the ISL takes to learn the user's behavior and achieve user satisfaction. This is because it starts learning the user's behavior from the best-matching behavior previously recorded rather than starting from scratch.

Our results suggest that over the experimental period, the embedded agent significantly reduced the user's need to intervene. This not only reduces the complexity of use but can bring significant cost and effort savings

The Authors

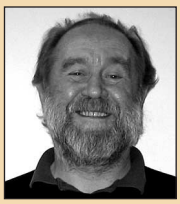


Hani Hagras is a senior lecturer in the Department of Computer Science at the University of Essex. His research interests include computational intelligence, notably fuzzy logic, neural networks, and evolutionary computation; ambient intelligence; pervasive computing; and intelligent buildings. He is a member of the IEEE and of the executive team of the IEE's Robotics and Mechatronics Professional Network. He received his PhD in robotics from the University of Essex. Contact him at the Dept. of Computer Science, Univ. of Essex, Wivenhoe Park, Colchester CO4 3SQ, UK; hani@essex.ac.uk.



Victor Callaghan is a senior lecturer in the Department of Computer Science at the University of Essex. He is also head of the Brooker Laboratory for Intelligent Embedded Systems and director of the Inhabited Intelligent Environments Group at Essex. His research addresses methods of integrating intelligence into embedded computers based on soft-computing techniques, notably fuzzy logic, neural networks, and genetic algorithms, aimed at pervasive computing and ambient-intelligence applications. He is a chartered engineer and a member of the BCS and IEE. He received his PhD in computing from the University of Sheffield. Contact him at the Dept. of Computer Science, Univ. of Essex, Wivenhoe Park, Colchester CO4 3SQ, UK; vic@essex.ac.uk.

Martin Colley is a lecturer in the Department of Computer Science at the University of Essex. His primary research interests involve developing parallel and distributed control architectures for robotics and intelligent machines, in particular for autonomous vehicles, and investigating how these architectures could benefit from interaction with their environment. He received his PhD in parallel and distributed computing systems from the University of Essex. Contact him at the Dept. of Computer Science, Univ. of Essex, Wivenhoe Park, Colchester CO4 3SQ, UK; martin@essex.ac.uk.



Graham Clarke is a computer officer in the Department of Computer Science at the University of Essex. His major research interests are in AI, embedded agents for intelligent inhabited environments, and object relations psychoanalysis. He received his MSc in computing applications from the University of North London and his PhD in psychoanalytic studies from the University of Essex. Contact him at the Dept. of Computer Science, Univ. of Essex, Wivenhoe Park, Colchester CO4 3SQ, UK; graham@essex.ac.uk.

Anthony Pounds-Cornish is a senior research assistant with the European Union-funded Social (Self-Organizing Societies of Connectionist Intelligent Agents Capable of Learning) Project. His main research interests are in computational intelligence, including online evolving of spiking neural networks using genetic algorithms, stigmergic communication, and embedded computing. He received his BSc in computer science from the University of Essex. Contact him at the Dept. of Computer Science, Univ. of Essex, Wivenhoe Park, Colchester CO4 3SQ, UK; apound@essex.ac.uk..

Hakan Duman is a senior research officer with the European Union-funded eGadgets Project. His main research interests are in computational intelligence, including fuzzy logic, neural networks, ubiquitous computing, and embedded agents. He received his BSc in computer science from the University for Applied Sciences in Regensburg, Germany. Contact him at the Dept. of Computer Science, Univ. of Essex, Wivenhoe Park, Colchester CO4 3SQ, UK; hduman@essex.ac.uk.

over the evolving lifetime of products by avoiding expensive programming (and reprogramming). In addition, it empowers ordinary users by letting them use collections of computer-based artifacts to design novel systems to suit their personal tastes without needing to understand the technical complexities or program the systems. The learning curve's shape in Figure 6 suggests that the agent moved increasingly closer to the user's environmental preference even though this preference was never static.

Our experiment also suggests that the agent requires surprisingly few rules to autonomously create a comfortable environment, with diminishing need for user correction. This is important new information: prior to this work, it was unknown whether a tractable rule set would emerge and whether embedded architectures with only megabytes of memory would be able to host agents for such ubiquitous-computing environments.

Our future experimental program includes plans for multiuser and multiroom habitation experiments. We also plan wider deployment of embedded agents (such as personal agents inside wearable technology) and experiments on differing agent granularities. We are building a multiroom version of the iDorm, called iDorm-2, as a preliminary step toward constructing a fully functional apartment (iFlat), which will house visiting researchers and act as a unique ubiquitous-computing test bed. We are currently designing the iFlat for such experimentation from the ground up. ⬒

Acknowledgments

We acknowledge the funding support of the European Union's Information Society Technologies Disappearing Computer program and the Korean–United Kingdom Scientific Fund program.

References

1. K. Ducatel et al., *Scenarios for Ambient Intelligence in 2010*, tech. report, Information Society Technologies Advisory Group (ISTAG), Inst. of Prospective Technological Studies (IPTS), 2001.
2. V. Callaghan et al., "Embedding Intelligence: Research Issues for Ubiquitous Computing," *Proc. 1st Equator Interdisciplinary Research Challenge Workshop Ubiquitous Computing in Environments*, Univ. of Nottingham Press, 2001, pp. 110–130.
3. H. Hagras et al., "Online Learning and Adaptation for Intelligent Embedded Agents Operating in Domestic Environments," *Fusion of Soft Computing and Hard Computing for Autonomous Robotic Systems,* C. Zhou, D. Maravall, and D. Ruan, eds., *Physica-Verlag*, vol. 116, Nov. 2002, pp. 293–323.
4. A. Bonarini, "Comparing Reinforcement Learning Algorithms Applied to Crisp and Fuzzy Learning Classifier Systems," *Proc. Genetic and Evolutionary Computation Conf.*, Morgan Kaufmann, 1999, pp. 52–60.
5. H. Hagras et al., "A Hierarchical Fuzzy Genetic Multi-Agent Architecture for Intelligent Buildings Learning, Adaptation and Control," *Int'l J. Information Sciences*, vol. 150, nos. 1–2, Mar. 2003, pp. 33–54.
6. A. Holmes, H. Duman, and A. Pounds-Cornish, "The iDorm: Gateway to Heterogeneous Networking Environments," *Proc. Int'l Test and Evaluation Association (ITEA) Workshop Virtual Home Environments*, ITEA Press, 2002, pp. 30–37.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.

IEEE Intelligent Systems

Call for Papers

www.computer.org/intelligent/cfp15.htm

Submission Guidelines: Submissions should be 3,000 to 7,500 words (counting a standard figure or table as 200 words) and should follow the magazine's style and presentation guidelines. References should be limited to 10 citations.

To Submit a Manuscript: To submit a manuscript for peer-reviewed consideration, please access the IEEE Computer Society Web-based system, Manuscript Central, at http://cs-ieee.manuscriptcentral.com/index.html.

Advanced Heuristics in Transportation and Logistics

Transportation and logistics organizations often face large-scale combinatorial problems on operational and strategic levels. In such problems, all possible combinations of decisions and variables must be examined to find a solution; consequently, no partial enumeration-based exact algorithm can consistently solve them. This occurs because sharp lower bounds on the objective value are hard to derive, thus causing a slow convergence rate. By exploiting problem-specific characteristics, classical heuristic methods aim at a relatively limited exploration of the search space, thereby producing acceptable-quality solutions in modest computing times. As a major departure from a classical heuristic, a *metaheuristic* method implies a higher-level strategy controlling a lower-level heuristic method. Metaheuristics exploit not only the problem characteristics but also ideas based on artificial intelligence rationale, such as different types of memory structures and learning mechanisms. Solutions produced by metaheuristics typically are of much higher quality than those obtained with classical heuristic approaches.

This special issue of *IEEE Intelligent Systems* will feature original, high-quality submissions that address all aspects of metaheuristic methods as applied to transportation and logistics. Applications-oriented papers will be extremely welcome, as well as papers addressing computational performance of metaheuristic methods on well-known benchmark instances.

Guest Editors

Christos D. Tarantilis, Athens University of Economics & Business
Diomidis D. Spinellis, Athens University of Economics & Business
Michel Gendreau, Université de Montréal

Submissions due 4 February 2005