Control and learning of ambience by an intelligent building

Ueli Rutishauser, *Member, IEEE*, Josef Joller, *Member, IEEE*, and Rodney Douglas

***Abstract*— Modern approaches to the architecture of living and working environments emphasize the dynamic reconfiguration of space and function to meet the needs, comfort and preferences of its inhabitants. Although it is possible for a human operator to specify a configuration explicitly, the size, sophistication and dynamic requirements of modern buildings demands that they have autonomous intelligence that could satisfy the needs of its inhabitants without human intervention. We describe a multi-agent framework for such intelligent building control that is deployed in a commercial building equipped with sensors and effectors. Multiple agents control sub-parts of the environment using fuzzy rules that link sensors and effectors. The agents communicate with one another by asynchronous, interest based messaging. They implement a novel unsupervised online realtime learning algorithm that constructs a fuzzy rulebase derived from very sparse data in a non-stationary environment. We have developed methods for evaluating the performance of systems of this kind. Our results demonstrate that the framework and the learning algorithm significantly improve the performance of the building.**

***Index Terms*— Intelligent buildings, ambient intelligence, multi-agent architecture, asynchronous messaging, online unsupervised learning, fuzzy logic**

I. Introduction

Buildings are changing their nature from static structures of bricks and mortar to dynamic work and living environments that actively support and assist their inhabitants. These new buildings are expected to behave intelligently. That is, the building is an active, autonomous, entity ([1], [2]) that pursues goals that relate to for example; energy consumption, security, and the needs of users.

Building intelligence poses a number of interesting challenges. Decisions must be made in near-realtime. The system must have a way to interact with its users to obtain feedback. On the other hand, it should not intrude on the user.

The user of a building should not have to interact directly with the building using special input devices: input should be derived from standard devices like switches and presence detectors that are available in any building. Specifying rules that describe which actions to take at which time because of which conditions is complex and time consuming. In addition, these rules have to be changed constantly, because preferences and needs of users change. It is neither convenient nor cost effective to program a sophisticated building manually, as this approach is too complex and static. The building thus needs to learn its own rules of behavior based on feedback it obtains from its occupants. Furthermore, it should continually adapt this knowledge.

We approach these challenges using a multi-agent control system that emphasizes local decision making. Each agent controls and learns about a small subregion of the whole state space.

Manuscript received September 2003; revised March 2004. This work was supported by the Institute of Neuroinformatics, Zurich, and the NSF Engineering Research Center for Neuromorphic Engineering (ERC) at the California Institute of Technology.

The authors are with the Institute of Neuroinformatics at the ETH/University Zurich (R.D., U.R.), the University of Applied Sciences Rapperswil, Switzerland (J.J.), and the California Institute of Technology (U.R.).

II. Description of the Problem

A building can be regarded as an intelligent agent that is itself recursively composed of other agents. Thus, on a conceptual level, intelligent buildings (IBs) are similar to other intelligent agents such as humans, animals and robots. This equivalence allows us to apply concepts and principles drawn from the broad literature of autonomous agents research ([1]) in the building task. However, IBs also have special requirements that require novel solutions, because unlike mobile agents that move through their environment, the building completely contains its environment that is composed largely of mobile agents (humans, robots, etc.) passing through itself.

A. Environment

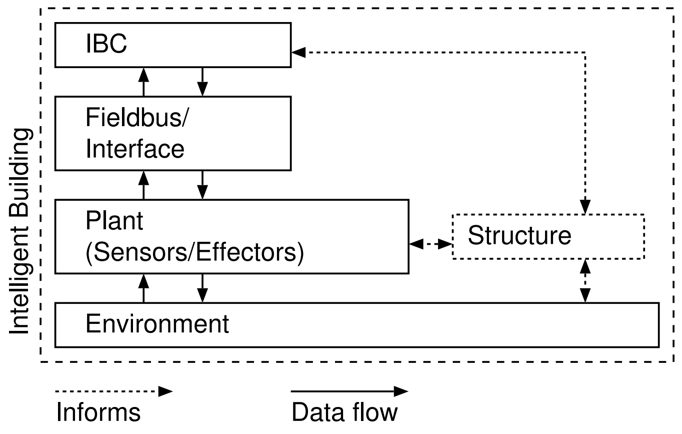

Fig. 1. Overview of our intelligent building. A multi-agent intelligent building controller (IBC) senses and controls the building environment via sensors and effectors attached to a common fieldbus. The IBC learns to control the environmental requirements of its users. The organizational structure of the building can be specified or learned.

Any kind of control and learning (figure 1) needs information about the environment (figure 2) that it is controlling. The less pre-specified the problem is the more must be learned

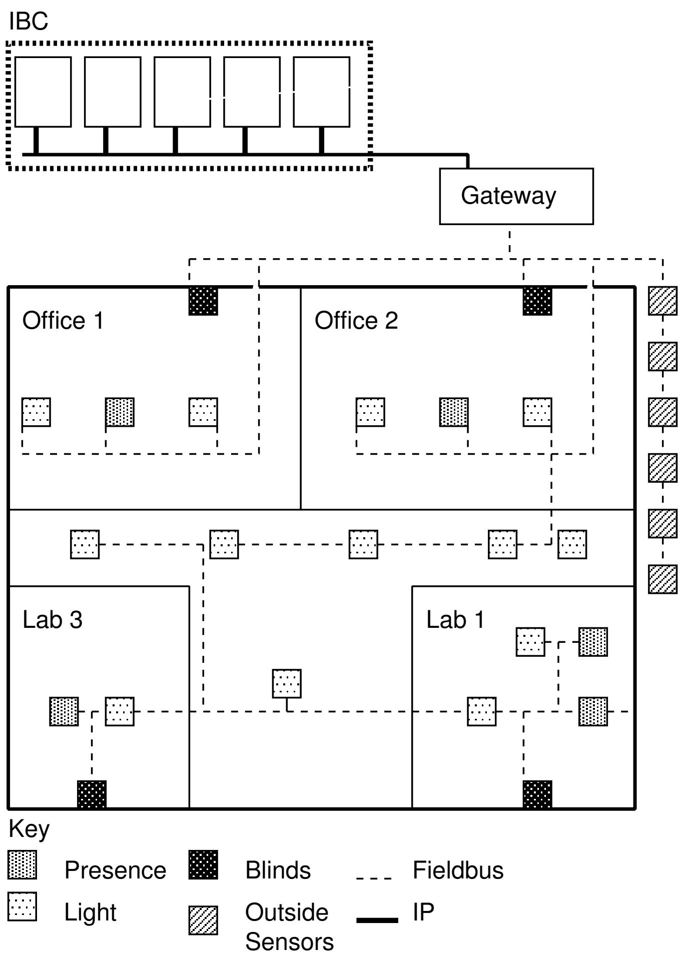

Fig. 2. The plant: A floor of a typical building structured into rooms. All sensors and effectors are wired to a common fieldbus network. A gateway allows the IBC to access the fieldbus network.

through interaction with the environment. However, less pre-specification also makes learning more difficult and it is more likely that the system will not be able to control its environment successfully. This trade-off between pre-specification and learning of the structure of the environment needs to be addressed.

Usually, learning occurs on the basis of a pre-defined representation (eg number of variables, possible values of variables), which implies that the underlying structure of the problem is static. Such stability cannot be assumed in the case of IBs, which must be able to detect and incorporate structural changes at any point of time. Changes could for example be new sensors and effectors (new variables) or additional physical structures that make previously related sensors unrelated (eg. a new wall). An additional question is what granularity the learning should have. Should it be on the scale of the whole building, a single floor, a room or even a single sensor/effector?

B. Representation of Knowledge

How can knowledge about the environment and the other agents that inhabit it be represented? Analyzing such a system is difficult and ill-defined because most of its interactions are with a real environment, which in practice includes many non-deterministic components. It is desirable to have a learning procedure that uses a human-readable representation (eg grammar, graph, decision tree, rules) that facilitates rapid incorporation of domain specific knowledge before learning begins rather than phenomenological methods like HMM's or Bayesian belief propagation networks.

C. Control and Learning

Implementing and maintaining decision rules manually for a dynamic environment such as an IB with its ever changing requirements is not possible in practice because of the time and effort required to continually re-write the rules. The ability to learn continually by itself ([3], [4]) is a basic requirement for an IB.

One of the major aspects of ambient intelligence is that such intelligence is *invisible*, in the sense that it operates without intruding on the user ([5]). Consequently an intelligent building does *not* have special mechanisms for interaction with its users. Instead, it should acquire all its data, including feedback for learning, from standard sensors (presence detectors, wall switches etc) in a normal building. This requirement implies that i) feedback for learning must be inferred from the state acquired from sensor data (eg is it reward or punishment?) and ii) that the feedback for learning is very sparse. Learning must be one–shot (one single sample must be sufficient to learn from). Because decisions and learning are expected to happen in realtime, learning must be online. Learning (training) and decision making happen simultaneously and so there can be no distinction between a training and a production phase.

D. Consistency of Goals

An IB should assist the user to make him more comfortable [6]. In this context *satisfaction* is defined to be minimization of user interactions: The less a user needs to instruct the building (eg. press a switch) to perform some action the more the IB satisfies the user. This definition implies that the user knows how to interact with the building, which is one of the main reasons why our system only relies on input from standard devices (see section II-C).

The demands of the human occupants are not the only demands that an IB must satisfy. The demands of non-human mobile agents (eg delivery or cleaning robots) and global constraints such as energy consumption or security must be met as well. These many demands change over time and may sometimes be contradictory because of changing usage of the building (new occupants) or changing preferences (non-stationary environment).

III. Approach

In the following sections we present a novel architecture and learning procedure that is capable of coping with the above requirements, and demonstrate the applicability of our approach by evaluating data accumulated from an extensive test period in a real building.

Our approach focuses on the building as a whole rather than on a collection of independent rooms. This approach is in contrast to others ([7],[8]), which focus on making a single room intelligent through the usage of special sensors and effectors like cameras or robotic arms. For example Coen et al. [8] detect where people are located in the room (person tracking), what these people are doing (pointing) and interact with people in the room by speech recognition (input) and speech synthesis (output). Our approach is different, because we regard the building as an entity that provides intelligence to the *ambience* without special devices or methods for interaction with the user.

Our system learns logic rather than statistics. Knowledge is represented by fuzzy rules that are used for decision making (see [9]). The fuzzy rules used for decisions are constructed by a learning process which continually modifies the set of rules according to feedback it receives from the environment.

In contrast to the approach by Hagras et al. [10], our learning algorithm is completely unsupervised. All feedback is acquired by means that the user does not notice, and by actions the user would also execute if there were no learning system present (e.g. switching on the light).

IV. Architecture

A. Multi-Agent System

Our IB is managed by a multi-agent system (MAS) in which there is no central coordinator. We define an agent (following [2]) to be *a system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives*. Depending on its sophistication, an agent may have one or several of the following properties: reactivity, pro-activity (goal-directed behavior) and social ability (interaction with other agents) ([2]).

Each agent is responsible for one specific task and offers this task as a service to other agents. Agents collaborate with each other to achieve their various aims. The collaboration is mediated by asynchronous messages.

Our system consists of several different types of agents (see figure 3). The agents of the lowest layer are responsible for the interface to the IB's device bus (see section IV-B.1). The middle layer consists of the *DistributionAgent* and *StructureAgent* (see sections IV-B and IV-C). The intelligent learning agents are located in the top level. These are various instances of *ControlAgent*, and are responsible for controlling the effectors.

All agents operate within the ABLE (Agent Building and Learning Environment) framework ([11]). ABLE implements JAS ([12]), an architecture for agents which run concurrently on physically distributed systems. JAS implements the FIPA abstract agent architecture (FIPA Standard 00001, [13]).

B. Interagent Communication

A MAS platform usually provides messaging services (eg JAS, [12]). The requirements for an interagent communication facility as we require it are different from those of traditional MOM (message oriented middleware). In contrast to traditional MOM, relationships between agents of a MAS are highly dynamic: New agents get started and running ones get stopped. Thus, a MAS communication facility does not know in advance which constellation of entities will make use of its services, or which types of messages will be transmitted.

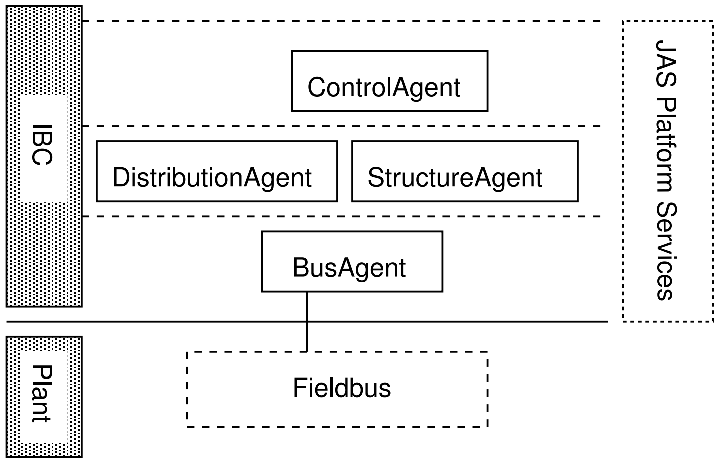

Fig. 3. The intelligent building controller (IBC) consists of multiple types of agents. The top layer is populated by many instances of the control agent; the middle layer contains the support agents (distribution agent and structure agent) and the lower layer the bus agent. The bus agent provides the interface to the plant.

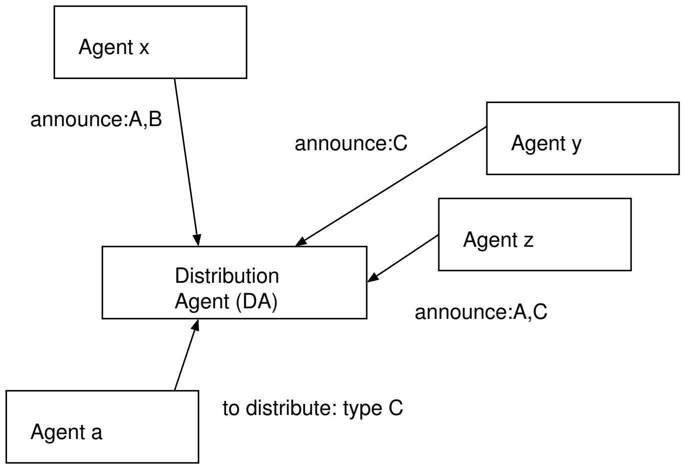

Fig. 4. Asynchronous, interest-based messaging between agents, which acts as a mediator between agents.

An agent that sends out a message of some type does not know which other agent is interested in receiving its message and does not wait for acknowledgment. This relationship between agents is managed by a *message distributor*, which is an independent agent running as part of the MAS. It collects the interests of all running agents and distributes messages according to those interests (figure 4). Agents announce their interests to the message distributor. An agent (agent a) wishing to send a message (type C) passes this task on to the message distributor, which distributes the message according to the collected interests of all the other agents. Distribution is done in parallel and asynchronously.

This and the fact that agents look themselves up in a directory makes a MAS loosely coupled and robust.

1) Topics: The DA (distribution agent) manages interests in terms of topics which represent a kind of message. All *register interest* and *unregister interest* messages relate to a specific topic. The DA does not know the available topics in advance: The first time an interest in a topic is registered it automatically creates this topic. If no agents declare interest in a topic anymore the distribution agent destroys the topic.

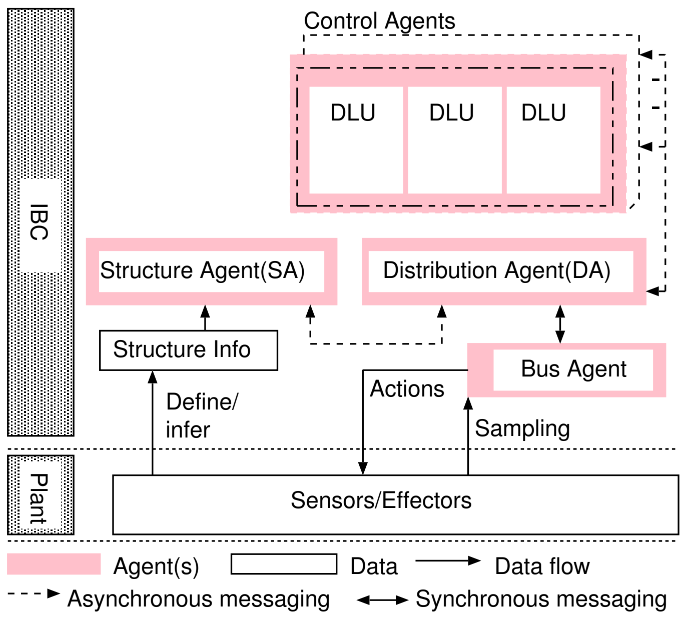

Fig. 5. Overview of interactions inside the MAS (upper part of figure, IBC) and with its environment (Plant, which is not part of the MAS). Multiple instances of the control agent are running, each of which instantiates one or multiple decision and learning units (DLUs, see figure 6 and figure 13).

Not only the communication between multiple agents is managed in terms of topics, but also the external data. External data describes all messages flowing between some agent that is part of the MAS and other entities (all physical sensors and effectors) which are not. This external data provides the interface between the environment and the MAS. An agent can effect the environment by sending a message to this specific topic (effector). Messages sent to an external topic are automatically forwarded to the agent responsible for external communication; In our case, the *Bus agent*. The bus agent is notified by the distributor whenever topics that represent external data are created or destroyed . The bus agent responds by starting/stopping delivery of data updates on these topics. Figure 5 summarizes the interaction between the agents and the environment.

C. Generic Structure Information

An intelligent building controller (IBC) requires extensive knowledge of the structure of the building it controls. This includes the relationships between sensors and effectors, as well as the relation of these effectors and sensors to the static structure (e.g. where are they physically located).In addition, such structural information must also be capable of relating dynamic and static elements, for example when a mobile agent moves through the building interacting first with one room, then with another.

Since structural information varies widely between different rooms, floors and buildings, it can not be directly encoded into the control system. Ideally, an adaptive system should not depend on pre-specified knowledge at all – it should be capable of discovering its structure itself. For this purpose our system uses a generic structure description service called *structure agent* (SA). All other agents request their structure information from this agent. The structure is described by a recursive composite structure similar to the composite pattern in object oriented software engineering ([14]). The description consists of clusters which contain other clusters and elements. Elements are either sensors, effectors, feedback devices or a combination of these. Figure 11 shows a visualized example of such a definition and figure 12 shows the formal definition of a single room. We choose a *cluster* rather than a room as the basic unit of building structure, because rooms often contain communities or regions of common interests that operate as a cluster. The clusters may even extend across the physical boundaries of rooms.

The definition of the structure can either be pre-specified (static), dynamically discovered or a mixture thereof. For simplicity we use a pre-defined static structure in this paper. We describe our approach for dynamic structure discovery from data by correlation analysis in Trindler et al. [15].

V. Control

The core of an IBC are entities which are taking decisions about the state of all effectors of the building. Such a system constantly evaluates all available data about the building and changes its outputs according to it. These decisions need to be taken in near realtime.

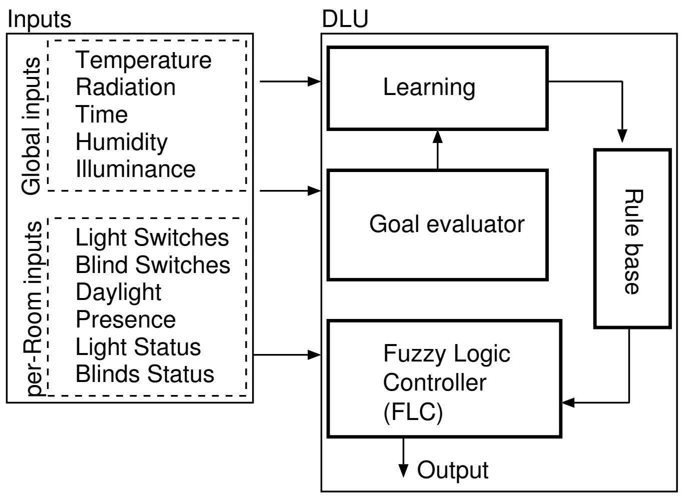

Fig. 6. A Decision and Learning Unit (DLU) consists of a fuzzy logic controller (FLC), a goal evaluator and a learner that generates rules from sparse events.

Our system takes decisions on the basis of a number of fuzzy logic rules (rulebase). A rulebase represents the knowledge of the outside world and specifies how to react to input signals. FLCs (fuzzy logic controllers), as shown in figure 6, constantly evaluate the inputs available and take decisions about the outputs of the system according to the fuzzy rulebase. The inputs available are, in our case, divided into two groups: The first group consists of inputs that relate to the building as a whole. This are variables such as humidity, temperature, radiation, illumination and time. The inputs of the second group are variables that are available for every room. These group consists of presence information, daylight (indoor light intensity), light status and blind status. The output of the decision making is binary, eg bring the blinds up/down and to switch on/off the light.

A. *Localized Decision Making*

The input and output of an IBC is high dimensional, yet decisions about particular lights and blinds are only influenced by a very small subset of all available data in the system. It is thus not practical (from a computational point of view) to use one single FLC to decide the state of all outputs of the system. To simplify the system from the point of view of the design and architecture, computational complexity, and learning we use a FLC for every output in the building. We call a unit which takes decisions and learns about one particular output *a decision and learning unit* (DLU, see figure 6). A DLU is capable of making decisions about it's output on the basis of very few input signals and is thus fast and efficient. This emphasizes the *localized decision making* approach of the overall architecture. Each instance of the control agent instantiates one or multiple DLUs (figure 5), depending on the number of outputs the cluster has that is associated to the control agent instance.

B. *Multilayer Decision Making*

Not all decisions are taken by agents. As many decisions as possible are made directly on lower levels to reduce system complexity and increase system stability. These purely reactive decisions are made directly by the fieldbus network, which binds for example all the connections between switches and the respective actuators. These bindings ensure that when a light-on switch is pressed, the appropriate light is turned on immediately. This minimum control allows the system to operate at a very basic level, even if the whole MAS fails. This is a very important requirement, as a building is something on which human occupants rely heavily.

C. *Fuzzy Rulebase*

The rulebase can either be engineered by a human expert or be produced by a machine learning algorithm. We take the later approach and use a fuzzy learning algorithm to learn the rulebase automatically on basis of a punish/reward online learning algorithm. The rulebase consists of a number of simple if-then rules:
R_1 : if x is A_1 and y is B_1 then z is C_1
R_2 : if x is A_2 and y is B_2 then z is C_2
R_n : if x is A_n and y is B_n then z is C_n

where R_n is the label of the rule, x and y are input variables and z is the output (control) variable; A_i, B_i and C_i are linguistic variables; x, y and z are fuzzy variables.

The rulebase consists of different types of rules: static rules and dynamic rules. Static rules encode the fixed requirements of the system that can not be changed, whereas dynamic rules encode the preferences of users. Dynamic rules are generated automatically on basis of feedback from the system by a learning process. In case of a conflict of interest between static and dynamic rules the decision taken by static rules gets preference. Only knowledge required by the IBC is encoded within the rulebase. All reactive decisions made on lower levels (fieldbus) are not part of the rulebase.

D. *Fuzzy Inferencing*

```
Fuzzy Temperature = new Fuzzy(-30.0 , 100.0)  {
 Shoulder A_veryCold = new Shoulder(-30.0, 0.0, L);
 Triangle B_cold = new Triangle(-5.0, 5.0, 14.0);
 Triangle C_warm = new Triangle(8.0, 16.0, 24.0);
 Triangle D_veryWarm = new Triangle(18.0, 33.0, 48.0);
 Shoulder E_hot = new Shoulder(28.0, 100.0, R);
};
```

Fig. 7. Formal definition of the fuzzy variable temperature (Celsius degrees) in ARL syntax whereas R is Right and L left.

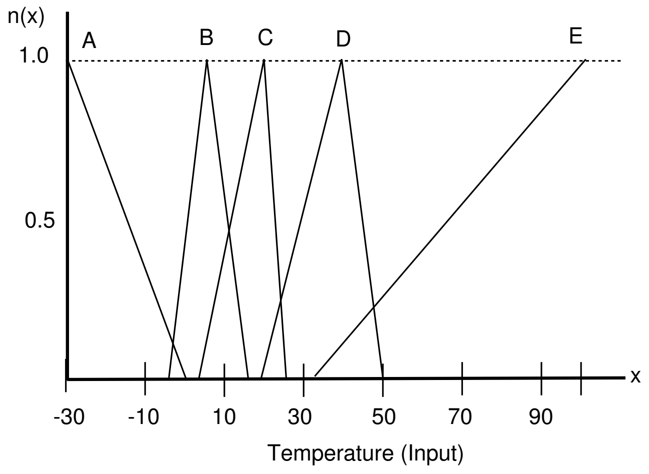

Fig. 8. Graphical definition of the fuzzy variable Temperature. It has membership functions labeled *A,B,C, D* and *E* as defined in figure 7.

The FLC as described above only uses fuzzy variables for computing its output (inferencing). All real-valued input data is first fuzzified with statically defined membership functions (Triangular, Shoulder and Trapezoidal [16]). Membership functions (see figure 7 and figure 8) are specified manually for every type of input data. These functions are pre-specified and are not learned. The membership functions incorporate implicit domain knowledge.

The steps executed by the inferencing engine are:

- fuzzification of input variables
- rule evaluation
- aggregation of the rule outputs
- defuzzification

We use mamdani-style fuzzy inferencing, and the center of gravity approach for defuzzification [16]. ARL syntax (Able

Rule Language, [17]) is used for the formal definition of fuzzy variables and rules, because it offers a simple, powerful, and Java-like syntax for the definition of fuzzy inferencing processes.

VI. Learning

Intelligent buildings pose a very difficult learning problem because they require online learning from sparse data in a non-stationary environment. Most machine learning algorithms are unsuitable for such an environment because they can not cope with non-stationary environments, online learning, or sparse data. Consequently, we developed our own learning algorithm specifically suited for such an environment.

A. Overview

Our algorithm extends the inductive fuzzy learning algorithm of Castro et al. [18], [19] and the work of Bonarini et al. [20], [21], [22]. It uses principles similar to inductive logic programming [23] to deduce logic from data. and produces maximal structure fuzzy rules that are continually adapted (driven by feedback from the environment, see figure 5). The maximal structure property ensures that at any point of time the state space is covered as completely as possible (e.g. rules generalize well). The algorithm copes with a non-stationary environment by not assuming that feedback from the environment is self-consistent.

Conceptually, the algorithm uses all available knowledge about the environment (samples) as constraints to construct a maximal structure rulebase. When a new sample becomes available this new knowledge is used to either strengthen or modify the existing set of rules. This principle is based on the premises of TMS (truth maintenance systems, [24]). A TMS is a theoretical approach commonly used to reason within the constraints set by all available knowledge about an environment [19].

Online learning requires a constant upper bound on memory usage to avoid the requirement of infinite memory in the limit of infinite running time. The bound means that it is not possible to store all available samples about the environment.On the other hand some memory of past samples must be retained so that new knowledge can be incorporated. We achieve this balance by storing only fuzzified samples.

Fuzzification of a sample involves the transformation of all real valued input values into semantic fuzzy labels. To do so, the fuzzification uses the definitions of the fuzzy variables (see section V-D). The number of all possible fuzzified samples is low (this constitutes the upper bound). A fuzzified sample can be regarded as a very specific rule (only a single data point in fuzzy state space). Every such data point p in state space has an output value y_p assigned to it.

All fuzzified samples together provide the constraints for the generalization process that consists of two operations: i) Fuzzy samples are merged together into single rules if they do have the same output value assigned to their respective data points in state space and ii) Fuzzy samples are enhanced (amplified) until they are as general as possible. This is done by gradually adding more and more conditions to the antecedents of the rules (See operators *amplify* and *amplifyIsPossible* in the following section).

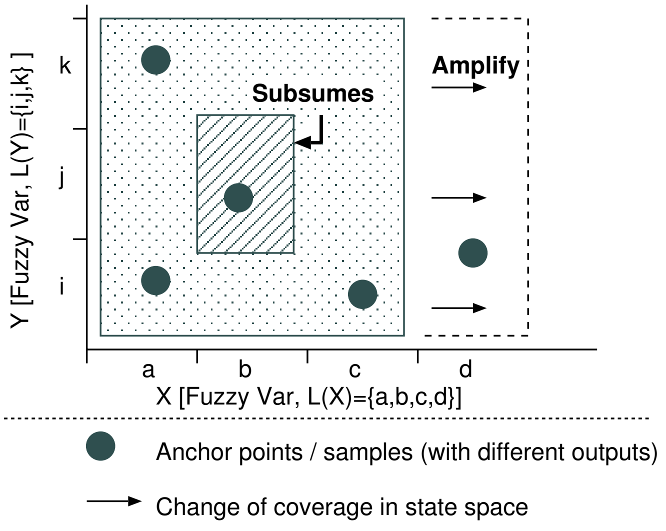
Y [Fuzzy Var, L(Y)={i,j,k}]
k
j
i
Amplify
Subsumes
a
b
c
d
X [Fuzzy Var, L(X)={a,b,c,d}]
Anchor points / samples (with different outputs)
Change of coverage in state space

Fig. 9. The effect of operators *Amplify* and *Subsumes* as shown in two dimensional fuzzy state space. Amplification adds additional conditions to the antecedents of a rule. A rule that subsumes another one is either more general than the other rule (both rules have the same output) or the rule contradicts the other one (contradictory output values).

New samples are triggered by events in the environment (change of state). The first action after fuzzification is to decide whether such a sample is a reward or a punishment, by deciding whether the output value associated with the input values of the sample contradict the current generalization or not. The operators *subsume* and *antecedentSubsume* are used (see next section for definitions and figure 9) to decide whether one rule subsumes an other one (eg includes an other rule, which then becomes irrelevant).

Because the environment is not stationary, it is not enough to only add samples to the known facts. Old ones must also be removed, if they contradict new facts. This detection is done by anchoring all generalized rules in the rulebase to known data points. When a rule is removed or modified in such a way that an anchor point is no longer part of the rule, then the associated sample is removed from the list of known facts. All valid samples are incorporated such that their constraints are met: There are no outliers.

B. Definitions

1) More General: A rule R_i is *more general* than rule R_j if it covers a bigger part of the state space than R_j.

2) Subsume: A rule R_i *subsumes* rule R_j if all antecedents of R_j are also antecedents of R_i and if the output of R_i and R_j are equal. *AntecedentSubsume* is true if only the first of the conditions for *subsumes* holds (but the outputs could be different).

3) Amplify: Amplify computes a set of rules which are generalizations of a rule R_i. *amplifyIsPossible* is either true or false for each of these generalizations. An amplification (a single rule) is possible if it does not contradict any facts (feedback from environment).

C. Algorithm

The following procedure is executed every time the system receives new input (which can be a punishment or a reward). Two sets are used: the *training set* $\mathcal{R}_T$ accumulates all fuzzified feedback received from the environment. The *definitive rule set* $\mathcal{R}_{DEF}$ contains the current set of rules (ruleset). $\mathcal{R}_{DEF}$ is the generalization of the constraints set by the feedback accumulated in $\mathcal{R}_T$. A rule r_T consists of a number of antecedents E_{Ti} such that $r_T = (X_1$ is E_{T1} and X_2 is E_{T2}.... and X_n is $E_{Tn})$, where X_i are fuzzy variables and E_{Ti} is a subset of the labels $\mathcal{L}(X_i)$ of the fuzzy variable X_i.
Algorithm:

0 Wait for new input.
1 Transform the real-valued sample into an training set entry (fuzzification). This new training set entry becomes rule r_T. r_T has a set of antecedent conditions E_{Ti} for every fuzzy input variable X_i.
2 Test for every rule r_D that is part of the definitive ruleset ($r_D \in \mathcal{R}_{DEF}$) whether antecedentSubsume(r_T, r_D) is true. If yes, assign $r_a = r_D$ ($r_a \in \mathcal{R}_{DEF}$) for the r_D for which antecedentSubsume(r_T,r_D) is true and go to step 3.
If no, go to step 8.
3 Test if subsume(r_T,r_a) is true. If yes, the input sample was a reward. Stop and go to step 0. If subsume(r_T,r_a) is false there is an invalid rule in $\mathcal{R}_{DEF}$ (environment changed). At this point the output value y_a of the sample is different from the output value y_T of a rule in the definitive ruleset ($y_a \neq y_T$). Go to step 4.
4 If all $E_{Ti} \subset E_{ai}$ $(i = 0..N)$ and a minimum of one of these subsets is a real subset ($|E_{Ti}| < |E_{ai}|$) the rule can be re-used. Go to step 5 in this case. If not, go to step 6.
5 Assign $r_m = r_a$. Reassign every E_{mi} as following: $E_{mi} = E_{ai} - E_{Ti}$ (remove all conditions $E_{Ti} \subset E_{ai}$ from E_{ai}). Add r_m to the definitive ruleset $\mathcal{R}_{DEF}$.
6 Remove r_a from the set of definitive rules $\mathcal{R}_{DEF}$ and go to step 7.
7 Test if there are entries in the training set $\mathcal{R}_T$ which became invalid because of the removal of r_a. Remove all training set entries T_j for which antecedentSubsume(T_j, r_a) is true but antecedentSubsume(T_j, r_m) is false. Go to step 8.
8 Add the new sample r_T to the training set $\mathcal{R}_T$. Go to step 9.
9 Amplify r_T and assign the resulting sorted set of amplifications to $\mathcal{R}_A$. r_{amp} is the rule currently considered ($r_{amp} \in \mathcal{R}_A$). Sort $\mathcal{R}_A$ such that the least general rule $r \in \mathcal{R}_A$ is the first element of $\mathcal{R}_A$ and the most general rule $r \in \mathcal{R}_A$ is the last element of $\mathcal{R}_A$.
10 Assign r_{amp} to the least general rule in $\mathcal{R}_A$ for which amyplifyIsPossible(r_{amp}) has not been called yet. Go to step 11.
11 If amplifyIsPossible(r_{amp}) is true, go to step 10. If false, add the last r_{amp} for which amplifyIsPossible(R_{amp}) was true to the set of definitive rules $\mathcal{R}_{DEF}$. Go to step 0.

VII. Evaluation, Experiments and Results

A. Experimental Setup

1) Building: The building for all experiments is a commercial office building located on the Irchel campus of the University of Zurich. One floor of this building, in which there are a number of offices and laboratories, is fully equipped with sensors and effectors that can be accessed via a serial field bus network (LON, see [25] for a discussion of LON and other competing field bus standards). Each room is equipped with one or several blind controllers, light controllers, light switches, blind switches and presence detectors. In addition to this there are outside sensors for weather data (illumination, radiation, humidity, temperature).

A typical room is equipped with 1-4 light switches, 1-2 blind switches, 1-4 light controllers and 1-2 presence detectors (figure 10).

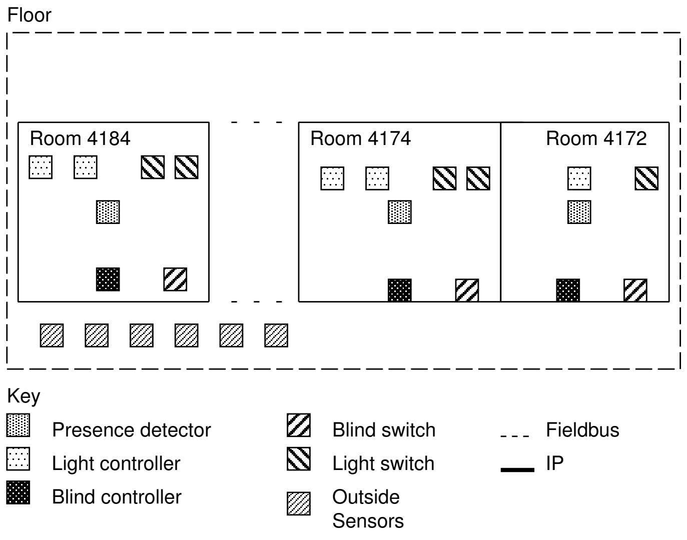

Fig. 10. Rooms from which data reported in this paper was obtained. Room number 4174, for example, has two light switches, two independent light controllers (each of these controls a number of lights), one independent blind controller, one blind switch and one presence detector. The presence detector also includes a luminance sensor.

2) Structure: Figure 12 shows the definition of the structure information for room 4184 as shown in figure 10. The definition of the structure consists of clusters and elements (figure 11). Every element is part of a single cluster and every cluster is part of an other cluster (recursively, except the root cluster). All decision making and learning takes place on the basis of clusters. All elements of type effector within the same cluster are affected by all elements of type sensor in the same cluster.

A single DLU manages each cluster that contains one or several effectors (eg effector=true in figure 12). Feedback for learning is acquired from elements of the cluster which are of type *feedback* (eg elements with feedback=true in figure 12).

3) Agents: Every control agent has an associated cluster that represents the root cluster for a particular room or parts of a room that it controls. For small rooms, one root cluster for each room exists; However, to efficiently control big rooms, we use several root-clusters that each control a subpart

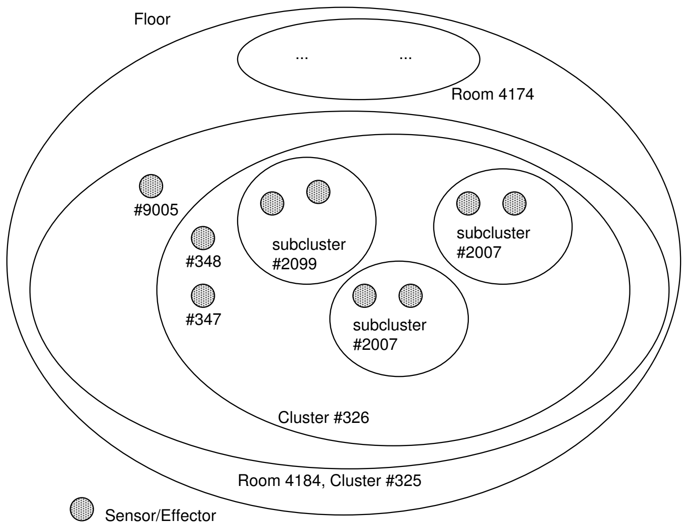

Fig. 11. Visualization of parts of the recursive structure of the building as defined formally in figure 12. One instance of the control agent is running for every cluster that represents a room, as is clusters #325. See figure 13 for details of the internal structure of the control agent for cluster #325.

```
<cluster id="325" displayName="4184_" >
 <cluster id="326" displayName="4184_" >
   <cluster id="2007" displayName="4184_">
     <element id="343" type="light"
     effector="true" sensor="true" feedback="false"/>
     <element id="345" type="lightSwitch"
     effector="false" sensor="true" feedback="true"/>
   </cluster>
   <cluster id="2099" displayName="4184_">
     <element id="344" type="light"
     effector="true" sensor="true" feedback="false"/>
     <element id="346" type="lightSwitch"
     effector="false" sensor="true" feedback="true"/>
   </cluster>
   [...]
   <element id="347" type="daylight"
     effector="false" sensor="true" feedback="false"/>
     <element id="348" type="presence"
     effector="false" sensor="true" feedback="false"/>
 </cluster>
 <element id="9005" type="info" effector="false"
 sensor="false" feedback="false">
   orientation=east
 </element>
</cluster>
```

Fig. 12. Structure information for a single room, expressed in XML syntax. ID #325 represents the room 4184 and consists of element #9005 and subcluster #326. #326 consists of a number of subclusters and elements (only some are shown). For every subcluster of #326 a decision and learning unit (DLU) is instantiated as shown in figure 13

(which operate independently). After start-up, the control agent dynamically discovers the structure of its environment (everything that is part of its root cluster) and creates a number of DLUs such that there is one DLU for every subcluster of the control agent's root cluster (see figure 13 for the case of room number 4184).

Every DLU is composed of a goal function evaluator, a learner and a fuzzy logic controller (FLC). The goal function evaluator asserts the interests of the system itself (eg saving energy, providing security) by providing feedback to the learner. The learner uses the feedback available, from the goal function evaluation or user feedback, to modify the rulebase. The FLC uses the rulebase constructed by the learner to take decisions. The sensors and effectors connected with a unit are defined by the structure information (figure 12).

One instance of the control agent manages each cluster that represents a room and thus n control agent instances are running if n is the number of rooms of the building. In addition a single instance of every of the following agents is running: distribution agent, bus agent, structure agent. That is, the number of agents in a running system is $n+3$.

4) Implementation: Running the system requires 3 standard PC servers. One of these servers is running the gateway between, in our case, the LON network and the higher level logic is running on the other two servers. These servers run the whole MAS. For stability and scalability reasons we choose to distribute the MAS over two physically different servers. One of the MAS servers is running the lower-level base services (messaging, bus abstraction) and the other one is running the higher level decision making and learning.

B. Analysis of Results

1) Input Data: Examples of raw input data sequences are shown in figure 14. See figure 6 for the input and output signals used.

2) Performance Evaluation: The performance for an IB could be measured in different contexts, eg energy, security, comfort or a weighted sum thereof.

Here we evaluate performance in the context of user comfort, postponing the issues of energy consumption and security. We define a measure *comfort* in the range 0...1, as the sum of all periods of time during which no user interaction was necessary compared to the total period of time. The highest performance possible means that there was no user interaction required. A period with no user interaction is defined as either:

- The period of time between two non-forced decisions, or
- The period of time between a forced decision (because of a punishment feedback signal) and a non-forced decision.

In periods of time between non-forced and forced decisions the user was not satisfied and they are thus not counted. Because the exact point of time at which the system state did not meet the needs of the user can not be defined exactly, the entire period is taken as the worst case for which conditions did not meet the wishes of the user.

$$c = \frac{1}{T}\Big(\sum_{i=1}^{|\mathcal{P}_N|} p_{Ni} + \sum_{j=1}^{|\mathcal{P}_F|} p_{Fj}\Big) \qquad (1)$$

$c \in [0 \ldots 1]$. Equation 1 is the definition of performance, $p_{Ni} \in \mathcal{P}_N$ is the set of all periods that start and end with a non-forced decision, whereas $p_{Fi} \in \mathcal{P}_F$ is the set of all periods that start with a forced decision and end with a non-forced decision. A period of time is defined as the number of seconds between two decisions (p_{Ni},p_{Fi} and T are in seconds).

To calculate the performance of a number of DLUs (for example a room or a floor) we take a weighted sum of all

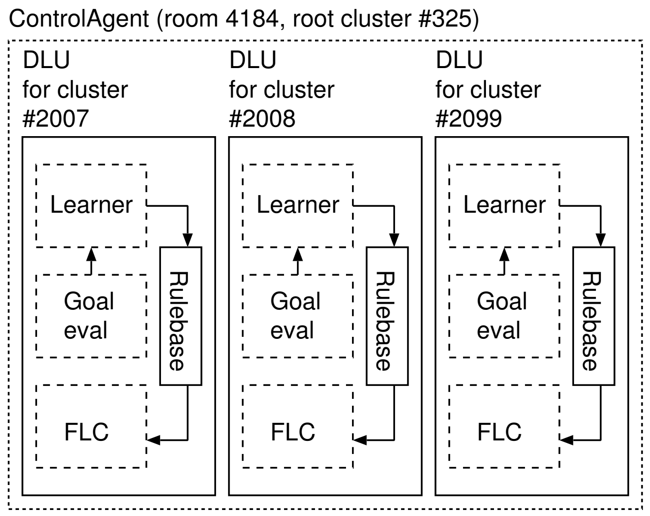

Fig. 13. A cluster of decision and learning units (DLUs) within the control agent. FLC: Fuzzy Logic Controller, Goal eval: Goal evaluator, DLU: Decision and learning unit.

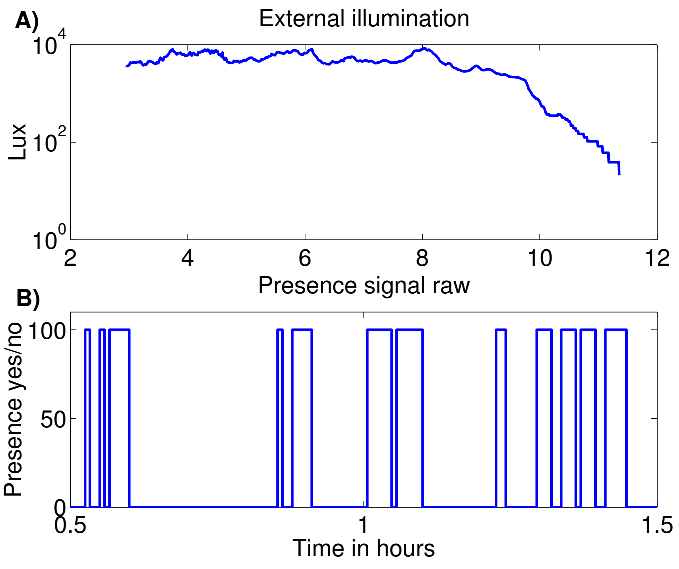

Fig. 14. Examples of raw data input: **(A)** External illumination (real valued), **(B)** presence (binary) is a series of binary pulses out of which the presence value is computed by a low-pass frequency filter.

comfort measurements and divide it by the total duration of the measurement period (eq 2).

$$c_{tot} = \frac{\sum_i T_i * c_i}{\sum_i T_i} \tag{2}$$

i is a specific DLU, T_i is the time since the start of learning of DLU i and c_i is the performance of DLU i as calculated with eq. 1.

Comfort is evaluated in relation to the total time elapsed since starting the experiment. This measure is useful in evaluating the short term performance of the system; but not its long term performance, because past comfort looses relevance to the user after a relatively short period of time. As the system continually adapts itself to new conditions performance may drop in regard to the past. To measure this fading relevance, we do not use the total time elapsed for evaluating the long-term performance, but rather use a running window within which c is calculated. A typical window size is $k = 70$ hours. That is, we apply equation 1 considering only periods of time $T(p_{Ni}) \geq t - k$ or $T(p_{Fj}) \geq t - k$ where t is the current point of time.

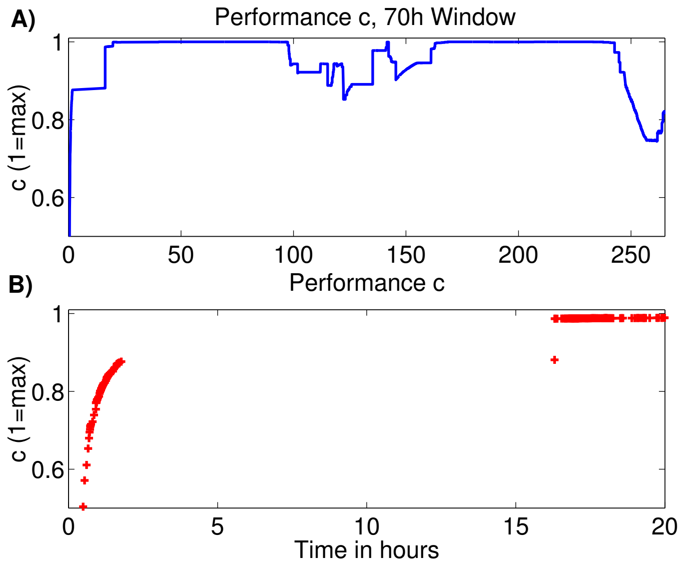

Fig. 15. Performance c of a single DLU using a 70h time window. **(A)** Overview over several days. **(B)** Zoomed in version of the same data during the first few hours of learning.

The comfort/performance c for a period of time for a single room is shown in figure 15. In figure 15A c is shown for a long period of time with a window size $k = 70h$. Figure 15B shows the first few ours of the same data in more detail. At point $t = 0$ learning started with a manually constructed default rulebase. At points $t > 0$ in both figure 15A and 15B it can be seen that the behavior of the system changed by learning and thus the performance changed. Note in figure 15A that performance drastically drops at about $t > 230$. This effect is due to change of preferences of a room occupant, which reduces performance for some time till the system adapts its rulebase (as shown in the case of $t = 150$).

3) Learning – Benchmark: Evaluation of online learning algorithms in non-stationary environments is difficult because there are no test-and training sets which could be used as a benchmark. It is not possible to construct such sets, because the learning occurs as a result of interaction with the environment and not as a result of observing the environment.

We compared the performance of our online algorithm with other algorithms using the *Iris dataset* [26] that is commonly used for such tasks. We used (randomly selected) 80% of the iris dataset (which consists of 210 samples) for training and 20% for testing. Castro et al. [19] used the same setting for an offline-version of our algorithm and report a correct classification rate of 93.3% ($\frac{28}{30}$). Using the same test and training set published in [19], our algorithm achieves the correct classification rate of 93.3% ($\frac{28}{30}$). This result is expected, because the online version should perform as well as the offline version (in the best case) provided that the training data is not self-contradictory. If the training data is self-contradictory (and sparse) the online version performs better.

4) Learning – Real Environment: Evaluating the performance of a learning agent that is running in a multi-agent system together with other learning agents is complicated by the fact that the environment in which the agents act is non-stationary, and that learning of all agents is online. Every agent (including those in the environment, eg humans) all influence the state of the environment.

We define a measurement R (eq 3), which we use for evaluating the success of a single instance of the learning agent:

$$R(t) = \frac{\sum_{i=j,d_j\in D_N}^{|D_N|} d_j}{\sum_{i=1,d_i\in D_F}^{|D_F|} d_i} \qquad (3)$$

where R is the ratio of all non-forced decisions taken by the system divided by all feedback signals (corrections) received by the system. A non-forced decision is a decision for which no correction was received within a reasonable period of time, whereas a forced decision is a decision enforced by feedback. A forced decision is not counted as an autonomous (independent) decision of the system. The bigger R, the better is the learning of the measured DLU. The measure c, by contrast, measures user satisfaction.

The following experiment was used to evaluate the success of the learning. Four rooms comprising 11 decision and learning units (with IDs 2010,2011,2012,2030,2031,3000,2090 and 2091) were used for the experiment. The rooms were all controlled for a period of 100 days. On day 1, all DLUs were started with a small default ruleset. On day 100, the system was shut down and the accumulated data were analyzed.

Only 9 of the 11 units were used for analysis. Two were rejected because they showed unusually good performance ($R > 60$), which might be due to errors or artifacts. Each unit made on average 889 decisions (range 242...1585, $\sigma = 600$) and received on average 124 punishments (range 51...287, $\sigma = 82$). The average ratio was $\langle R \rangle = 7.2$.

The accumulated number of forced (manual interactions) and non-forced decisions for four different DLUs are shown in figure 16. The increasing difference between forced and non-forced decisions is the effect of learning. In the first few days both the number of forced and non-forced decisions are similar, but later on learning contributes and the number of non-forced decisions increases relative to the number of forced decisions (manual interactions). The larger number of IBC decisions reflects the fact that the IBC is responding to changes in the environment also in the absence of users. Figure 17 shows how R increased over time. This is because the number of system decisions relative to the number of manual interactions increases due to learning. The average absolute number of decisions and manual interactions are depicted in figure 18, which demonstrates that the absolute number of system decisions increased while at the same time the absolute number of forced decisions remained constant or decreased.

For every output an average of 1.24 feedback signals were received and an average of 80 decisions taken every day.

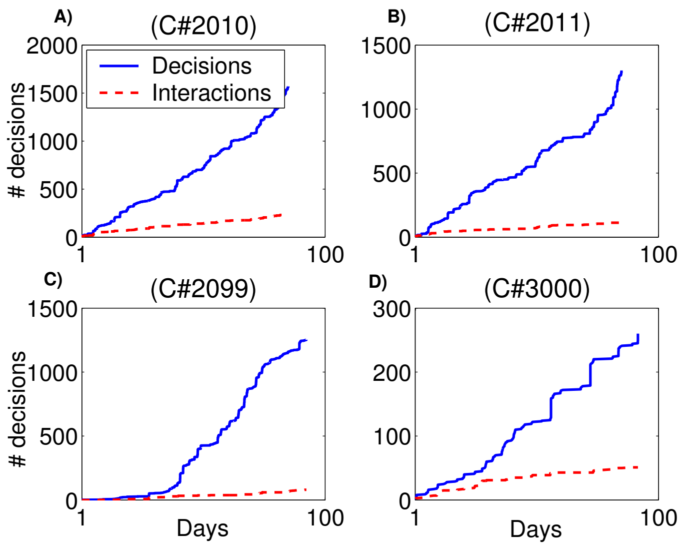

Fig. 16. Accumulated number of interventions and non-forced IBC decisions. The increasing difference between the IBC decisions and user interventions is due to learning. **(A)**, **(B)**, **(C)** and **(D)** are for learning and decision making units 2010, 2011, 2099 and 3000 over a period of about 100 days.

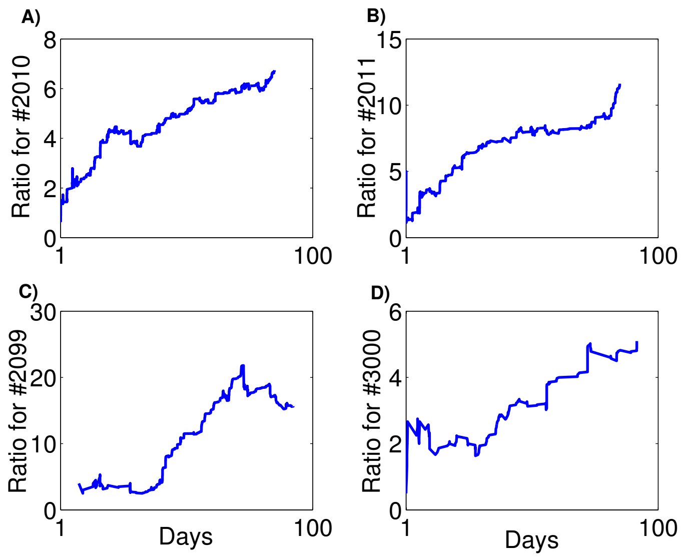

Fig. 17. Ratio of the number of decision taken by the system in relation to the number of user interventions. A ratio $R = 1$ means that all decisions taken by the system were wrong, ie the user reversed the system's decision on every occasion the building acted. A ratio $R > 1$ means that the system contributes. **(A)**, **(B)**, **(C)** and **(D)** show R for the learning and decision making units 2010, 2011, 2099 and 3000 for a time period of about 100 days.

Thus, the average number of punishments per hour was $\frac{1.24}{24} = 0.0155$, which demonstrates the sparseness of the data.

The results show that our algorithm succeeds in learning the dynamics of the occupants of the IB. Transient decreases in R confirm that the system is capable of recovering from inconsistencies caused by conflicting data. However, the ratio R sometimes decreases or stays constant for a long period (Figure 17). These phases are due to false or invalid feedback (e.g. someone experiments with the system or makes a mistake). The present system incorporates every sample into the rulebase immediately, and so well established long-

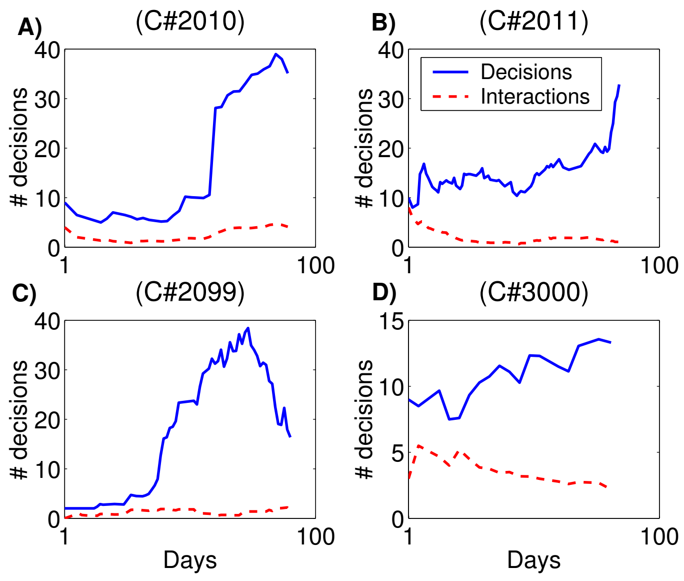

Fig. 18. Average absolute number of interventions and non-forced IBC decisions in a 24h period using a 15 day window. **(A)**, **(B)**, **(C)** and **(D)** are for learning and decision making units 2010, 2011, 2099 and 3000 over a period of about 100 days.

term knowledge could be degraded or destroyed by a single misleading sample. In these cases it may take some while for the system to recover to its previous level of generalization. In future versions of the algorithm we will introduce short-and long-term knowledge ([27]) to address this stability-plasticity dilemma [28].

VIII. Conclusions

We have described the organization and operation of an IBC that consists of multiple agents. The agents communicate with one another by asynchronous, interest based, messaging. To facilitate decision making and learning in realtime, each agent only observes and takes decisions about a small part of the environment.

Decisions are taken on the basis of a set of fuzzy rules which represent the knowledge of the system. There are two groups of rules: static and dynamic ones. Static rules establish fixed boundaries for the system whereas dynamic rules are learned and modified continually. Our learning algorithm constructs the fuzzy rulebase online and unsupervised, from sparse data that is acquired from the non-stationary environment.

We defined measures to asses the ability of the IBC to satisfy its users. Using these we showed in a real building that the IBC's performance improved over time, in that relatively more decisions are taken by the building than by the user.

Acknowledgment

The authors would like to thank the anonymous reviewers for there helpful reviews and our collaborators and colleagues at the Institute of Neuroinformatics and the California Institute of Technology for their help, support and advice. We would like to thank Prof. Hava T. Siegelmann (Univ. of Massachusets at Amherst) for valuable discussions and Alain Schaefer (Univ. of Applied Sciences Rapperswil, Switzerland) for implementing parts of the software.

References

[1] G. Weiss, Ed., *Multiagent Systems, A Modern Approach to Distributed Artificial Intelligence*. Cambridge, Massachusetts: MIT Press, 1999, ISBN : 0-262-23203-0.

[2] M. Wooldridge, "Intelligent agents," in *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, G. Weiss, Ed. Cambridge, MA, USA: The MIT Press, 1999, pp. 27–78.

[3] M. Mozer, "An intelligent environment must be adaptive," *IEEE Intelligent Systems*, vol. 14, no. 2, 1999.

[4] ——, "The Neural Network House: An Environmnet that Adapts to its Inhabitants," in *Proceedings of the American Association for Artificial Intelligence Spring Symposium on Intelligent Environments*, M. Coen, Ed. Menlo Park, CA: AAAI Press, 1998, pp. 110–114.

[5] N. Shadbolt, "Ambient Intelligence," *IEEE Intelligent Systems*, vol. 18, no. 4, pp. 2–3, 2003.

[6] H. Hirsh, "Roomservice, AI-Style," *IEEE Intelligent Systems*, vol. 14, no. 2, pp. 8–19, 1999.

[7] M. Coen, "Building brains for rooms: Designing distributed software agents," in *Proceedings of the Ninth Innovative Applications of Artificial Intelligence Conference*. AAAI Press, 1997.

[8] ——, "Design principles for intelligent environments," in *Proceedings of the 1998 National Conference on Artificial Intelligence*. AAAI Press, 1998.

[9] K. Eng, D. Klein, A. Baebler, U. Bernadet, M. Blanchard, M. Costa, T. Delbruck, R. Douglas, K. Hepp, J. Manzolli, M. Mintz, F. Roth, U. Rutishauser, K. Wassermann, A. Whatley, A. Wittmann, R. Wyss, and P. Verschure, "Design for a brain revisited: The neuromorphic design and functionality of the interactive space Ada," *Reviews in the Neurosciences*, vol. 14, no. 1–2, pp. 145–180, 2003.

[10] H. Hagras, V. Callaghan, M. Colley, and G. Clarke, "A hierarchical fuzzy-genetic multi-agent architecture for intelligent buildingsnext term online learning, adaptation and control," *Information Sciences*, vol. 150, no. 1-2, pp. 33–57, 2003.

[11] J. P. Bigus, D. A. Schlosnagle, J. R. Pilgrim, W. N. M. III, and Y. Diao, "ABLE: A toolkit for building multiagent autonomic systems," *IBM Systems Journal, Applications of Artificial Intelligence*, vol. Vol 41, no. Nr.3, pp. 350–371, 2002.

[12] Fujitsu Laboratories, "JAS Agent Services (JSR-87) Specification," 2002. [Online]. Available: http://jcp.org/en/jsr/detail?id=87

[13] "FIPA Abstract Architecture Specification (XC00001K)," Foundation For Intelligent Physical Agents, Geneva, Switzerland, Tech. Rep., 2002.

[14] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995, ISBN : 0-201-63361-2.

[15] J. Trindler, R. Zwiker, U. Rutishauser, J. Joller, and R.Douglas, "Discovery of logical structures in a multisensor environment based on sparse events," in *Proceedings of Workshop on Ambient Intelligence, 8th National Congress of Italien Association for Artificial Intelligence, Pisa*, 2003.

[16] R. Fuller, "Neural fuzzy systems," in *Advances in Soft Computing Series*. Berlin/Heildelberg: Springer-Verlag, 2000, ISBN : 3-7908-1256-0.

[17] *ABLE Team, Able Rule Language (ARL) Documentation*, IBM Thomas J. Watson Research Center, 2002.

[18] J. L. Castro, J. J. Castro-Schez, and J. M. Zurita, "Learning maximal structure rules in fuzzy logic for knowledge acquisition in expert systems," *Fuzzy Sets and Systems*, vol. 101, pp. 331–342, 1999/2/1.

[19] J. L. Castro and J. M. Zurita, "An inductive learning algorithm in fuzzy systems," *Fuzzy Sets and Systems*, vol. 89, pp. 193–203, 1997/7/16.

[20] A. Bonarini, "Anytime learning and adaptation of structured fuzzy behaviors," *Adaptive Behavior Journal*, vol. Vol 5, no. Nr. 3-4, 1997.

[21] ——, "Evolutionary Learning of Fuzzy rules: competition and cooperation," in *Fuzzy Modelling: Paradigms and Practice*, W. Pedrycz, Ed. Norwell, MA: Kluwer Academic Press, 1996, pp. 265–284.

[22] ——, "Delayed Reinforcement, Fuzzy Q-Learning and Fuzzy Logic Controllers," in *Genetic Algorithms and Soft Computing, (Studies in Fuzziness, 8)*, F. Herrera and J. L. Verdegay, Eds. Berlin, D: Physica-Verlag, 1996, pp. 447–466.

[23] M. Lavrac and S. Dzeroski, *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.

[24] J. D. Kleer, "An assumption-based TMS," *Artificial Intelligence*, vol. 28, pp. 127–162, 1986.

[25] D. Snoonian, "Smart buildings," *IEEE Spectrum*, vol. 40, no. 8, pp. 18–23, 2003.

[26] R. Fisher, "The use of multiple measurements in taxonomic problems," *Annual Eugenics*, vol. 7, pp. 179–188, 1936.

[27] H. Zhou, "CSM: A Computational Model of Cumulative Learning," *Machine Learning*, vol. 5, pp. 383–406, 1990.

[28] S. Grossberg, "Adaptive pattern classification and universal recoding: I. parallel development and coding of neural feature detectors," *Biological Cybernetics*, vol. 23, pp. 121–134, 1976.

Ueli Rutishauser is a PhD Student in Computation and Neural Systems at the California Institute of Technology in Pasadena, CA, USA. He holds a BS in Computer Science from the University of Applied Sciences Rapperswil, Switzerland.

Josef Joller is a professor in computer science at the University of Applied Sciences (HSR), Rapperswil, Switzerland. He studied mathematics and physics at the Swiss Federal Institute of Technology in Zurich, Switzerland, computer science at German Universities and conducted research at several universities and research labs. He obtained a Doctorate in theoretical Physics from the University of Bochum. Professor of Computer Science at HSR since 2001.

Rodney Douglas is Professor of Neuroinformatics, and Co-Director at the Institute of Neuroinformatics of the Swiss Federal Institute and the the University of Zürich. He graduated in Science and Medicine, and obtained a Doctorate in Neuroscience at the University of Cape Town. Thereafter he joined the Anatomical Neuropharmacology Unit in Oxford, where he continued his research on the anatomy and biophysics of the microcircuitry of cerebral together with Kevan Martin. As Visiting Associate, and then Visiting Professor at Caltech, he extended his research interests in neuronal computation to the modeling of cortical circuits using digital methods (together with Christof Koch), and also by the fabrication of analog VLSI circuits (together with Misha Mahowald). In 1996 he and Kevan Martin moved to Zurich to establish the Institute of Neuroinformatics. In 2000, Douglas was awarded the Körber Foundation Prize for European Science.