Adaptive Building Intelligence

Multi-Agent Framework to Administer and Control a Commercial Office Building

Jonas Trindler, Raphael Zwiker, Joseph Joller, Rodney Douglas

trindler@ini.phys.ethz.ch, rzwiker@ini.phys.ethz.ch

University of Applied Sciences, Rapperswil and Institute of Neuroinformatics, University / ETH Zürich

I. Introduction

Current research is concerning with the integration of autonoumus intelligence into our everyday life. Many attempts are trying to interact, improve user comfort and provide security in modern working and living envrionments. We present a software framework that is able to administer, manage and control a typical commercial office building. It acts with the environment through common devices like lights, window blinds, etc. and senses from illumination-, temperature-, radiation-, daylight-sensors and presence detectors. The integration of any other devices is possible (mobile devices, virtual computer network or household aids).

The framework serves as a fundamental component for any further analysis of a complex and non-stationary environment. It features numerous highly flexible, reliable and general services to access and control devices of a building on an abstact level.

Input systems

Most commercial buildings have an integrated building bus system (like LonWorks or EIB) with several well-known devices like lights, blinds, switches, presence detectors and many other sensors. Often is the building indeed equipped with more than one bus systems, which control e.g. the heating system or administer the power management. The framework must therefore be able to handle more than one different bus system simultaneously. It is also thinkable to connect the framework to virtual software busses like e.g. instant messaging systems (ICQ) or an ethernet bus.
These buses operate in their own special way and it is therefore absolutely necessary to abstract these various bus types with a common access interface.

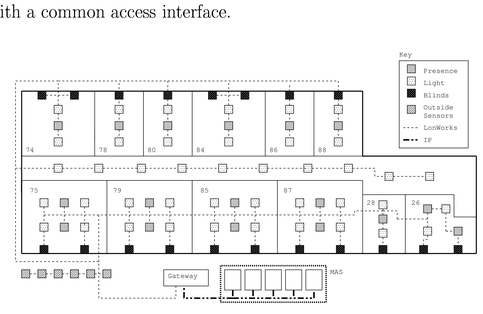

Figure 1: *Structure of the controlled building*

Demands and Consistency of Goals

In a commercial building are working different people who will all have a different perception of an optimal working or living environment. They would like to maximize their comfort as much as possible whereas the management tries to minimize the running costs and the caretaker wants to use devices with consideration. The framework should offer a possibility that each inhabitant and involved party can reach his objectives or otherwise find a trade-off to maximize satisfaction in general.

Access to the physical environment

The framework implements up to now a controller to access devices on a LonWorks bus system from Echelon. An IP-LonWorks-Gateway provides us access to the LonWorks network from on IP-based network. The gateway itself is composed of an *iLon-Gateway* and a *LNS-Server*.

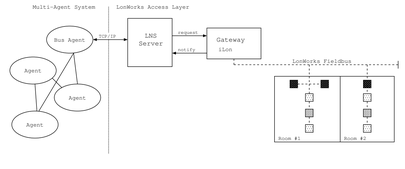

Figure 2: *Concept how our framework is able to connect itself to the LonWorks fieldbus from Echelon.*

II. Multi-Agent-System (MAS)

For a better understanding and to breakdown the complex problem domain, we want to divide the software into small software units. Each of these software units is responsible for one specific part of the whole problem. Conventional software design concepts use interfaces and classes to reach a low coupling, but they are indeed still one software unit.
We design each of these software units as an agent. An agent itself is a stand-alone software and is able to collaborate or interact with other agents. For example an energy agent can be designed to enforce claims of the management to save energy. Or a comfort agent can provide maximum comfort to the inhabitants.

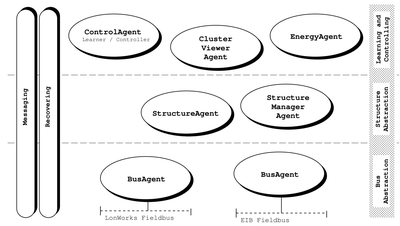

Figure 3: *Dependency of the agents as a layer model. Messaging and Recovering is used of every agent.*

III. Core Services / Core Agents

The core services of our framework provide the basic functionality for any higher level agent. They are realized with four different agents:

Bus Agent

The Bus Agent abstracts a specific hard- or software bus system with a common access interface. The framework starts for each specific bus type an individual *BusAgent* with the specific access implementation. Thereby our framework is able to control effector devices of different bus types at the same time. There are currently two implemantation of the *BusAgent*, one for the LonWorks fieldbus from Echelon and another for a virtual software bus (PC bus).

Structure Agent

Since structual information varies widely between different rooms, floors and buildings it can not directly be encoded in a software unit. Especially in modern working and living environments the structure is in a constant change. Ideally, an intelligent system should not depend on pre-specified knowledge at all. The *StructureAgent* maintains the structure of a building with all its devices and users. It abstracts all devices offered by *BusAgents* and holds them in a recursive composite structure.

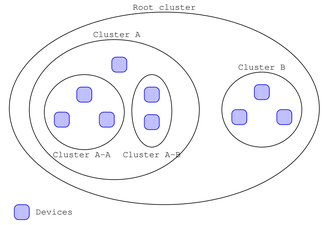

Figure 4: *The StructureAgent combines devices to cluster. The clusters itself are hierarchically arranged.*

The structure can be requested by all other agents from the MAS. The structure itself is dynamic and can be modified at runtime whereas structure updates will be sent through asynchronous messages to all interested agents.

Message Distribution Agent

The *MessageDistributionAgent* (MDA) provides an asynchronous, interest-based notification system similar to observer pattern which is well known in object-oriented software design. The system is comparable with the concept of mailing lists. Each agent can create or subscribe itself to a topic channel on the MDA and post messages to it. The MDA distribute the posted message to all subscribed agents whereby the sender is not aware of the number of subscribers.

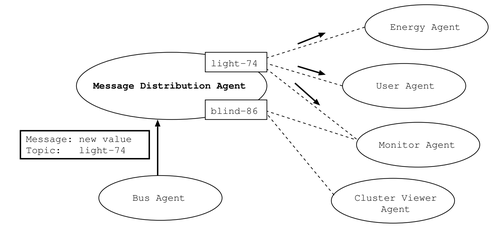

Figure 5: *Concept of the MessageDistributionAgent. The received message for topic 'light-74' is distributed to all agents which are subscribed to this topic.*

Recovery Agent

The *RecoveryAgent* provides the reliability of our framework. It prompts periodically each running agent to save its current state into a persistent state. With these continously made backups, all agents of the system can restore itself to a previous state without loosing too much knowledge due to a system crash or power blackout. The implementation of the storage und recovery is part of each agent itself.

IV. Control Agents

On top of the core services it is possible to run several control agents which are trying to control effectors based on its own goal. At the moment, we implemented two different agents where one is representing a normal user and another takes care about the reduction of energy consumption.

- **User Agent:** The *UserAgent* learns the behavior of a user and controls the effectors based on the learnt knowledge. The effective learning and controlling algorithm is implemented in a specific controlling unit (CU) with a general interface which enables that each *UserAgent* can run with another CU.
- **Energy Agent:** The *EnergyAgent* has only one global target: to save energy. This Agent observes the devices of a cluster and triggers to switch off all lights or set up all window blinds based on pre-specified rules (e.g. if nobody is present, switch off all lights).

V. User Interaction Agents

There are several additional graphical agents, which provides access to the framework to administrate the structure, monitor the system or simply control effector devices.

- **Cluster Viewer Agent:** With the *ClusterViewerAgent* the user gets the possibility to observe the devices its states within a cluster and to control all effectors also structured in this cluster.
- **Structure Manager Agent:** The *StructureManagerAgent* is able to add and remove devices or clusters. So the user can administrate the current structure with an easy graphical interface.
- **Personal PC Agent:** The *PersonalPCAgent* combines the switches of all effectors within the current cluster in a graphical control panel. The user can control the lights or the blinds directly from his computer. In addition, we use this agent also as a personal persence detector.

Code Statistic: 109 classes, 35 packages, 21'353 lines of real code

© December 2003; Documentation / Source Code is availabe at: http://www.trjo.ch/projects/abi/