HAT Architecture

by
Attilio Giordana
DISIT, Università del Piemonte Orientale
attilio.giordana@uniupo.it

Abstract

This paper describes a platform implemented in C++, called HAT, which provides an agent based architecture for implementing domotic applications. Agents are basically Linux processes, which interact by exchanging messages. The benefits deriving from this type of architecture are essentially three. Firstly, the agent paradigm provides a good abstraction mechanism for implementing modular systems, easy to configure according to the requirements of a specific application. The second reason is the possibility of scaling up to complex applications exploiting parallel and distributed computing on low power micro-pc. The third reason is load balancing and fault tolerance. When required, the agents migrate from one host to another providing fault tolerance and load balancing capabilities. HAT is provided with a self-configuring protocol, inspired to *zeroconf*, which makes plug-and-play installing it on a local network. Agents communicate among them via a restful protocol over http and are integrated in a web environment which offers an ubiquitous access from any kind of portable device, while providing a secure access. HAT is now a mature product, which cover most typical applications in the home automation domain, such as lite and socket control, anti-intrusion, and energy management. Nevertheless it is also a good starting point for quickly developing from scratch new functionalities.

1 Introduction

What is an agents
How agents can cooperate in order to realize complex applications
Modularity, self-configuration, redundancy, scalability.

This paper describes a distributed system, designed for automation, named HAT (Home Agent Team). The characteristic of HAT, which distinguishes it from other commercial systems designed for the same task is the agent based architecture, which is in the line of [1], [2]. This choice is due to several different reasons. The first one is modularity. Every agent implements a specific function, and is a self-consistent building block. Depending on the user needs a site dependent network of agents is defined in order to implement the required functions.

The second reason is parallel and distributed computing. The hardware suitable for home automation is usually based on low power CPUs, which may be installed in a wall-box, but do not provide enough power for handling in real-time complex events. Exploiting parallel and distributed computing is a solution that allows computing intensive applications be implemented without scaling up to more expensive hardware.

A third reason is fault tolerance [3]. Several home applications, for instance anti-intrusion, are critical with respect to fault tolerance. A failure in the system may endanger things and human beings. Serveral copies of a same agent may exist on different hosts providing redundancy. The control migrate from one to another copy in case of failure.

HAT's agents are implemented in C++ and run as Linux processes distributed on hosts interconnected by a TCP/IP local network.

Agents cooperate by exchanging messages encoded according to a restful protocol. Every agent is multi-threaded in order to be more reactive to concurrent events.

2 Multi-agent Approach

HORUS is implemented as an overlay network of agents distributed on a TCP/IP network of hosts providing a UNIX compatible platform like Linux or MacOS. More specifically, hosts may belong to two categories:

- General purpose mini/micro-pc, offering a full UNIX environment;
- Custom devices like IP-videocameras and IO-boards, which provides http based interface and are accessible as web sites.

  The agents are allocated on the general purpose hosts and communicate over http using an xml based protocol. Custom hosts, are then included in the agent community as *special* agents. The agent communication protocol is extensible in order to include the characteristic idioms of the custom hosts. An example of agent network including IO-boards and IP- videocameras is provided in Figure 1.

2.1 Forward agent network

2.2 Three layer architecture
virtual devices, manager, web interface

2.3 Restful interface

2.4 Self-Configuration

3 Agent Architecture

All agents have a similar architecture organized as an acyclic forward graph of threads communicating by means of message queues. The reason for choosing this architecture is to avoid possible dead-locks or delays on critical sections. Exam- ples of agents are reported in Figure 2 and 3, corresponding to an MD-agent and to a manager, respectively. More specifically the WEB-Thread provides the http interface to the incoming message from other agents, and is able to immediately answer without delays. The Trigger thread, accounts for the time flow and schedules internal actions at prefixed intervals. The Core thread is agent specific and implements the real work carried by the agent. Finally, the Destination and the Action threads are in charge of forwarding event, alarm, and action messages, to a list of other agents.

The Archive thread, which is presents only in some agents like MD-agents and REC-agents, is in charge of storing in

the archive the videos acquired from the cameras. Finally, the Filter thread, visible in the MD-agent architecture, is the one executing the video analysis checking for events corresponding to items moving in the scenario. Finally, the Log thread (to not be confused with the LOG-agent) provides to the other threads a logging service, on a file, for debugging purposes.

log-Thread

WEB-Thread

Command (set status)

Timer

archive- Thread

des1na1on- Thread

Trigger Thread

core-Thread

Filter-Thread

Scenario changes Motion detected

Fig. 2. message queues. As an example, the figure describes an MD-agent

Fig. 3.

The manager agents are provided with a rule knowledge base.

Every agent is organized as a set of threads communicating through

log-Thread

WEB-Thread

Event

Command (set status)

Timer event

Alarm

Command (action)

Rule Knowledge Base

des1na1on- Thread

core-Thread

ac1on-Thread

Trigger Thread

3.1 Multi-thread architecture

3.3 Thread interaction

3.2. Rest Interface

4. Agent Types

5. Web Interface

6. Conclusion

Bibliography

1. C.-L. Wu, C.-F. Liao, and L.-C. Fu, “Service-oriented smart-home architecture based on osgi and mobile-agent technology,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, vol. 37, no. 2, pp. 193–205, 2007.
2. W. Shen, Q. Hao, S. Wang, Y. Li, and H. Ghenniwa, “An agent-based service-oriented integration architecture for collaborative intelligent manufacturing,” *Robot. Comput.-Integr. Manuf.*, vol. 23, no. 3, pp. 315–325, Jun. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.rcim.2006.02.009.
3. S.Kumar,“The adaptive agentarchitecture: Achieving fault-tolerance using persistent broker teams,” in *In Proceedings of the Fourth International Conference on Multi-Agent Systems.* IEEE Computer Society, 2000, pp. 159–166.