Heterogeneous Device Networking for an AmI Environment

José Jaime[1], Cristina Urdiales[1], and Francisco Sandoval[1]

ISIS group, ETSI Telecommunications, University of Malaga, 29071 Malaga, Spain
jariza@uma.es, {cristina,sandoval}@dte.uma.es

Abstract. Assisted living environments involve a wide range of different devices. Most of them are commercially available, but typically associated to standard domotics buses not compatible with each other. Besides, in many cases it is desirable to integrate new devices to a system that might not support the installed bus protocol. Interconnection between devices is far from simple, specially because domotic buses are often proprietary. The most popular solution to this problem is to export information to Ethernet as a system meeting point, but it is not always simple and accessibility in proprietary buses is limited. This paper proposes a method to integrate a variety of platforms through a shared memory interface, including a proprietary bus, commercial devices and ad hoc systems. Its main novelty is that compatibility between different standards is achieved without additional expensive hardware.

Keywords: Domotics, KNX, Ambient Intelligence, protocols, integration.

1 Introduction

An Ambient Intelligence (AmI) environment is one that can raise context awareness through the use of sensors embedded in everyday objects. Whereas in traditional domotics systems do not need major intelligence, e.g. smoke and gas sensors or HVAC (Heating, Ventilation, and Air Conditioning) systems, AmI systems require further smartness, but also connectivity, so decisions can be made regarding all available information. Hence, AmI devices are networked and control can be performed in a distributed, hybrid way.

Many AmI systems are typically built on top of a domotics system, where several sensors are embedded in usual domestic elements like lights, smoke alarms, switches, presence detectors, etc. There are many commercial domotics protocols, both open, like X10 or BatiBUS and proprietary like Lonworks or KNX. Every bus has its typical niche depending on factors like how many devices it can support, its reliability or its cost. In many AmI applications it can be interesting to mix different buses for several reasons: i) some non critical devices are significantly cheaper in less reliable buses like X10; ii) critical devices like smoke or gas alarms might not be available in less reliable buses; iii) it might be necessary to combine different medium accesses. Besides, it is often interesting to include

in our systems other elements that do not have a domotics bus interface, like biometric sensors, robots, computers, etc. However, most protocols are meant to operate as a standalone installation, where each device is preprogrammed via commercial hardware to work in a deterministic way. This problem can be solved via a gateway that taps in the bus and make it available to external systems. While there are hardware gateways to provide partial access to the bus, they are usually meant for tele control and limited to activating/deactivating a device or group[1]. Other gateways may provide translation up to layer 2 or higher[2] to allow communication via USB, IP (KNXnet/IP) or even web services; in some cases KNXnet/IP to webservice conversion relies on external middleware software. The main drawback of IP communication is a significant protocol overhead; as all connection oriented protocols, TCP needs more process than non connection oriented ones. This is specially relevant in domotics networks, where messages are small and scarce. This could be partially solved by using UDP instead of TCP, but the overall overhead remains high. If communication relies on a web service, the overhead is even larger: there is XML over HTTP over TCP over IP[1], and XML and HTTP are quite heavy. Most of these web services use REST instead of SOAP to reduce the load [2], but it is still large. Alternatively, the middleware can run on a local computer using a light communication protocol, whereas web services run in remote ones [3], but, as most systems in a smart house are remote, web service usage would be high.

Web services have another drawback: since domotics buses operate on interruptions, the middleware needs to be able to both send and receive push notifications. This implies a duplicity in implementation, as both ends need a webserver and a webclient to work both ways. Finally, we need a translation system to abstract the inner addressing in the bus so that the user can send orders or receive status in a transparent way. This translation, when implemented, tens to be poor and doesn't extract all the information that every KNX message has [4].

This paper focuses on integrating a set of proprietary bus devices and other external devices in a common framework to implement an AmI system, while avoiding the problems commented above. In our case, we will focus on the KNX-EIB standard because it is widely available and supports almost every sensor and actuator in the market. Next section focuses on describing the protocol.

2 Proposed distributed architecture

In order to bring all possible devices together, we are going to use the DLA (Distributed and Layered Architecture)[3], originally proposed in [5] to control autonomous robots. Robotics (open) architectures are interesting for AmI applications because they support mechanisms to cope with physical agents and sensors in potentially unpredictable environments that fit well the AmI paradigm.

[1] http://www.cdinnovation.com/page1.html
[2] http://www.knxshop.co.uk/catalog/Catalog.aspx?NavID=000-009-1041
[3] Available at (GPL): http://webpersonal.uma.es/de/eperez/

DLA relies on a shared memory schema at local level, avoiding most protocol overhead. Remotely, although DLA is also based on sockets, it is much lighter than webservices and does not need to duplicate servers for push notification because its management server handles interruptions.

The main advantages of DLA are that it supports intuitive adaptation to different physical agents and simple expansion of their capacities via addition of new modules. This feature is specially interesting because it allows addition of different sensors and actuators in the same way that we could add a new robot with its own set of sensors. DLA combines the responses of different deliberative and reactive algorithms through the interaction of freely distributed processes in an asynchronous way. This is particularly interesting to implement the AmI paradigm of Ubiquitous Computing, where different devices may present different computing capacities and processors may also be distributed in the network.This architecture provides transparency to the user through a high simplicity and portability. Besides, it has a very low computational load. There are alternatives to DLA, specially in the robotics field. ROS (Robotic Operating System) is probably, the most popular one, but there are others like OpenRDK, OROCOS or MARIE. However, most of them are very oriented to robotic applications and neglect integration of domotic devices. Besides, they usually provide a full software framework to operate. While this means that they support a wide variety of hardware and software, the require more computational resources than DLA, that can run perfectly on devices like, for example, a Raspberry Pi. Besides, most frameworks tend to favor a specific OS. ROS, for example, runs on Ubuntu Linux and it is reported to have limited support on Mac OSX, but it is not easy to configure on Mac. DLA supports Unix, Linux and Windows alone or combinedly and devices simply need an API library to connect to the system. It also supports applications in C, Java and Matlab and can easily be extended to any programming language supporting sockets. Finally, regarding scalabitity, we can improve the system processing capability by adding new processing units on demand and extend functionality but progressively adding new equipment and behaviors to the system. We have already used DLA in different robotic platforms to achieve autonomous behaviors [6][5]. In this work we are going to add two new capabilities to a DLA based system: i) interaction with standalone hardware; and ii) interaction with a KNX bus.

2.1 Accessing the KNX bus

KNX is based in the EIB protocol stack but extended with several capabilities from BatiBUS and EHS. KNX is the de facto European standard and, although it is expensive when compared to open bus standards, it is within acceptable margins for domestic use. Its main advantages are:

- Interoperability: (KNX-compatible) devices from different manufacturers can be mixed; allowing high flexibility level installations.
- Quality: To use the KNX brand, manufacturers have to fulfill ISO 9001 and EN50090-2-2 (European standard for home and builds electric systems).

– Specific functionalities implementation: manufacturers that share the same interests can proposed the inclusion of new functionalities into the standard.
– Agnostic platform: KNX can be developed into any kind of hardware or software platform, even from already existing commercial products.
– Devices variety: KNX is supported by most manufacturers and offers a wide catalogue of device, including critical alarms.

The bus can be accessed in a standardized way through the BCU (Bus Coupling Units), which can be programmed, configured and commanded via the KNX bus. There are commercial software packages that show information on the installation state and alarms, usually via a graphic interface, but most don't support M2M (Machine to machine), so they are not very interactive.

Alternatively, there are some APIs that allow enhanced access to the bus like Falcon[4], Windows-only official API, or Calimero[5], Java-based API but unable to connect through KNX/USB adapter. In our case, we are going to use a set of free, Linux-based tools with the capacity of accessing through KNX/IP or KNX/USB and programming and commanding the BCU: BCUSDK [7]. BCUSDK provides a daemon for bus access, EIBD, that provides interaction with KNX via UNIX sockets or even TCP/IP (with reduced capabilities). Interfacing with the daemon is much easier than understanding the full protocol itself.

2.2 Integration of KNX into DLA

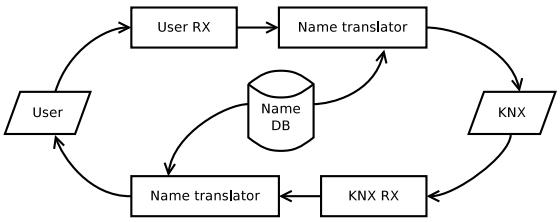


Fig. 1: Gateway functional schema

In order to interact with a KNX installation, we need to send commands to the devices and receive their status Fig. 1. We have two interfaces: i) communication with KNX, via EIBD using a UNIX socket; and ii) communication with DLA (shared memory system). Our system must perform a name translation from KNX group and device address. Our gateway runs two independent threads:

– sender, takes commands from DLA, translates the names into KNX group address and and sends the requests to EIBD. This thread is blocked until the user sends a command to KNX.

[4] http://www.knx.org/knx-tools/falcon/description/
[5] http://sourceforge.net/p/calimero/wiki/Home/

```
if send to KNX then
    groupaddr = GetFromTable(T1, 'group addr WHERE
    name=data[destname]');
else
    name = GetFromTable(T2, 'name WHERE source addr=data[source] AND
    group addr=data[group]');
    if name == '' then
        name = GetFromTable(T3, 'name WHERE source addr=data[source]');
    end
end
```

Algorithm 1: Name lookup process where data is the information to be send or the information received respectively.

– receiver, monitorizes communications in KNX. Every time there is a telegraph flowing in the KNX bus, it is read by EIBD and send to the gateway. The telegram sender address is translated into a name and notified to DLA. This thread is blocked until EIBD sends any notification.

All telegrams sent to KNX are also read by the receiver thread, to log the user commands, in case we want to implement some kind of data verification. The gateway understands three commands: i) WRITE sets the value of a parameter of a KNX element, and is used to control the domotic elements; ANSWER is generated when any KNX device send a telegram; and iii) READ ask a device to resend its status to the bus. It's used for forcing updates. The name translation (Alg. 1) is supported by three lookup tables (Table 1) and performed in 2 steps:

1. If the user wants to send a command to the KNX bus, it uses an address group, so we need a lookup table to translate names into it (T1).
2. If a device sends a telegram, we have a source address (device address) and a destination address (group address). There are two possibilities depending on how relevant the destination address is:
   (a) Relevant: the same binary input can send orders to several actuators. We can figure out from the destination address which input has been activated depending on our design. The system has a lookup table (T2) that combines both addresses to perform name translation.
   (b) Irrelevant: several inputs from the same binary adapter send orders to the same actuator. We can't know which input has been activated, since the information is duplicated. Hence, we use a third lookup table (T3) to relate source address with names.

All lookup tables are stored in a MySQL database. When the gateway receives a telegram from the KNX bus, all tables are queried and results are logged and used to track the devices status. When the user sends a command to KNX, only T1 is queried to translate the name in the command into a group address. The gateway also tracks the status of every element in the bus.

Since all the data needed for the name translation is stored in MySQL tables, devices can be added or removed in run-time by simply adding or removing the requiered row in the lookup tables.

T1		T2		T3	
Column	Type	Column	Type	Column	Type
id	int(11)	id	int(11)	id	int(11)
group addr	varchar(15)	source addr	varchar(15)	source addr	varchar(15)
name	varchar(100)	group addr	varchar(15)	name	varchar(100)
		name	varchar(100)		

Table 1: Database schemas for lookup tables T1, T2 and T3

3 A test scenario

We have tested the proposed integrated system in different scenarios. The following one consists of responding to a fire, and it follows the 112 emergency system recommendations[6]. We present this one because it involves all the commented interconnected systems, namely:

- A commercial home robot (Pioneer AT), with audioconference capabilities.
- A KNX basic installation with several sensors, actuators and alarms.
- A multiparametric physiological monitor (Equivital from Hidalgo).
- Wearable RFID reader for object and location detection.

Our KNX installation includes access to lights and blinds, plus information on the different rooms status via motion sensors. It can also cut off power input via a shut trip releaser. The key element for this scenario is a smoke detector that is not available in all domotic standards, but fairly common in KNX. In our specific case, it was purchased from Jung.

Whenever the smoke detector is triggered, KNX exports the information to the DLA system, that dispatches three parallel tasks: i) notify caregiver, ii) KNX actions; and iii) biometric monitoring:

- According to recommended safety protocols, commands are directed to the KNX bus to enhance visibility by turning on all the lights and also to close all the blinds to prevent air flows and, consequently, fire propagation.
- If the user is wearing the biometric sensor, it start to monitorize the user health parameters to check if he has already started suffering from smoke intoxication.
- The authorized caregiver is notified of the problem.

The caregiver may choose to contact the user and check if everything is all right. At their command, the system can send the robot to the room where the KNX system detected the user last. The robot detects where it is and where the user is within the room by means of a set of passive RFID tags set around the house and worn by the user Fig.3a. The robot carries around the reader itself. Both the reader and the robot are connected to the DLA architecture. When both robot and user are close, an audio conference is enabled to check the status

[6] http://www.112.es/consejos/incendio-de-un-edificio.html

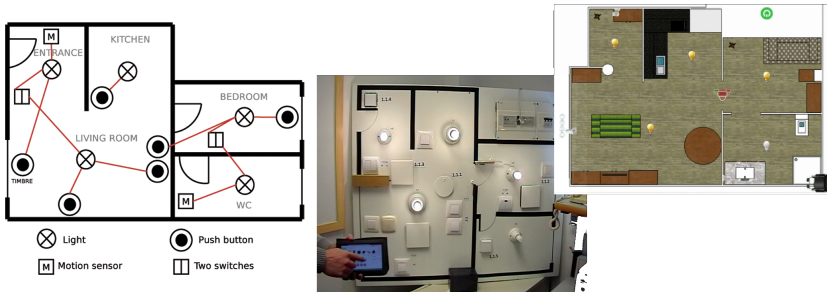
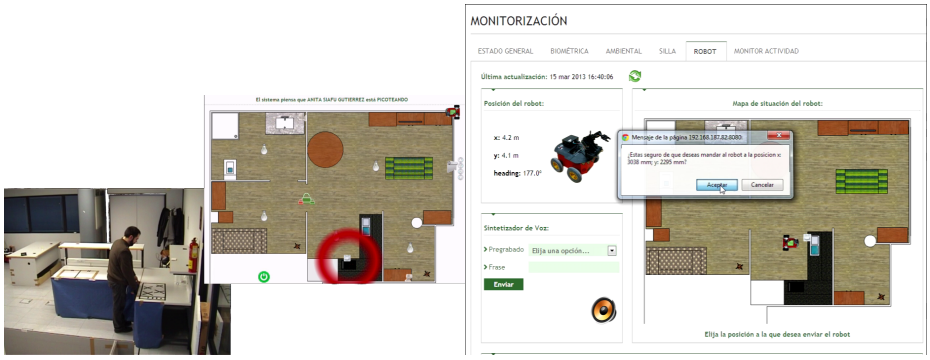


Fig. 2: KNX installation, simulation panel and user interface



(a) (b)

Fig. 3: a) RFID tag detection a) robot targeting and producing audio warnings

of the emergency and provide instructions to leave the house Fig. 3b. It is up to the caregiver to call the firemen and health services, because the system can not do it automatically due to current legislation. The user of the system can do it as well, but in case of emergency early evacuation is advised. The user's progress through the house is followed by the KNX motion sensors and the RFID tags reader. When the system detects that the user has left the house, the KNX system cuts off the electricity input Fig.2, according to safety protocols.

If the user cannot reach the main door, the caregiver can lead him via robot audioconference to the nearest window, which is automatically opened via a KNX command, and warn firemen about the situation. The location of the user within the house is constantly updated as long as the system is on-line.

The system was throughly tested in a controlled environment where all physical sensors were deployed. All name translation possibilities exposed in the previous section were tested. We checked that the system responded as expected in every test scenario we tried (flood, medical emergency, fall, etc) and that all devices shared information in a transparent way to the emergency protocol programmer.

4 Conclusions

This work has presented an architecture to integrate a KNX bus, a robot and different assistive and monitorization devices into an AmI architecture for emergency management. The main advantage of this heterogeneous approach is that the developer can choose the most appropriate device for each task, rather than focusing on a single standard or resigning to whatever (limited) information commercial interfaces offer via the usual web interface. The gateway developed is based on EIBD daemon and is able to understand all the non-programming commands. Our translation tool provides total access to the KNX bus and shows how it can interact with any other devices.

Future work will focus on extending the capabilities of the system from emergency management to profile construction, so that we can also generate alarms when the user deviates from his/her usual activities of daily living (ADL).

5 Acknowledgements

This work has been partially supported by the Spanish Ministerio de Educacion y Ciencia (MEC), Project n. TEC2011-06734 and by the Junta de Andalucia, SIAD Project No. TIC-3991. It has been also granted by Universidad de Málaga, International Campus of Excellence Andalucía Tech.

References

1. Kastner, W., Szucsich, S.: Accessing knx networks via bacnet/ws. In: Industrial Electronics (ISIE), 2011 IEEE International Symposium on. (2011) 1315–1320
2. Bovet, G., Hennebert, J.: A web-of-things gateway for knx networks. In: Smart Objects, Systems and Technologies (SmartSysTech), Proceedings of 2013 European Conference on. (2013) 1–8
3. Cruz-Sanchez, H., H.L., Chehaider, M., Song, Y.Q.: Mpigate: A solution to use heterogeneous networks for assisted living applications. In: Ubiquitous Intelligence Computing and 9th International Conference on Autonomic Trusted Computing (UIC/ATC), 2012 9th International Conference on. (2012) 104–111
4. Nazabal, J.A., M.I.F.V.C.F.F.B.P., Mukhopadhyay, S.: Home automation based sensor system for monitoring elderly people safety. In: Sensing Technology (ICST), 2012 Sixth International Conference on. (2012) 142–145
5. Pérez Rodríguez, E.J.: Distributed intelligent navigation architecture for robots. AI Commun. **21**(2-3) (April 2008) 215–218
6. Poncela, A., U.C.P.E., Sandoval, F.: A new efficiency-weighted strategy for continuous human/robot cooperation in navigation. Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on **39**(3) (2009) 486–500
7. Kögler, M.: Free development environment for bus coupling units of the european installation bus. Master's thesis, Institut für Rechnergestützte Automation, Technische Universität Wien (2011)