XVII IMEKO World Congress
Metrology in the 3rd Millennium
June 22 −27, 2003, Dubrovnik, Croatia

REMOTE TRACING OF IN-HOUSE EVENTS IN HOME AUTOMATION

Kay Werthschulte, Friedrich Schneider

Technische Universität München, Institute for Measurement Systems and Sensor Technology, Munich, Germany

Abstract - Measurement and control systems are essential in industrial applications. They are used to control and automate all kinds of processes during production. Their use in private homes is not so obvious due to the fact that the requirements are totally different. Although there are systems to connect sensors and actuators inside buildings using fieldbus technology they are only used for simple tasks like switching light and controlling the heating installation.
Through a gateway it is possible to build tele-services that allow any kind of maintenance by the inhabitants or service personal. The service personal will be able to run diagnostics on parts of the system as far as allowed (heat control, failure analysis on certain devices, surveillance, etc.). The access has to be restricted to prevent unauthorized persons to intrude the system.
This work will describe the gateway and the concepts used to share well defined services that are provided to users remotely and locally.

Keywords: Remote maintenance, services, home-automation.

1. INTRODUCTION

This work is based on results taken from a research project called "*tele*-Haus" sponsored by the German Federal Ministry of Education and Research (BMBF). For that purpose a building near to Munich was built which is equipped with components based on the European Installation Bus (EIB), ISDN and an Ethernet network. The project's aim is to built new components using concepts of modular micro systems. Besides developing these components another focus was to benefit from the network infrastructure in form of extending the control by tele-services.

2. SYSTEM ARCHITECTURE

In general there are several bus systems which can be used for in-house communication. The focus of this work should be set to home automation mainly. In this context there are three kinds of applications for bus systems: multimedia, (tele-)communication and HVAC (Heating, Ventilation, Air Condition). There is at least one bus system which could be taken into concern for usage: IEEE 1394 for multimedia, Ethernet for communication and EIB for HVAC appliances only to name a few. The demands and characteristics of these bus systems are quite different. Multimedia applications are demanding a guaranteed high data rate. IEEE 1394 uses an isochronous transfer mode to deal with voice and video streaming without repeating any information in case of data loss. In contrast to that Ethernet is a packet oriented communication media which does not offer isochronous data transfer modes. In conjunction with TCP-UDP/IP Ethernet is ideal for communication services. The control of HVAC does not require an isochronous data link. The main task is to provide a basis to transfer small units of data in a secure way.
All bus systems can be categorised by bus speed, kind of data links (e.g. isochronous, asynchronous, connection less, connection oriented) and their meaning for certain applications. Coupling these bus systems requires physical and logical connections. The physical connections are built by using gateways. The gateway concept includes future extensions and support for not yet considered bus systems.

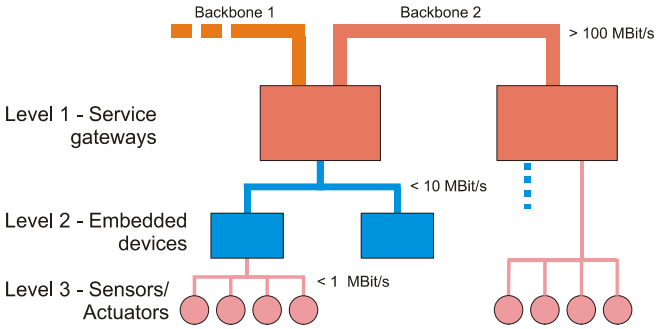

Fig. 1: Hierarchical physical system structure

The structure refers to similar approaches known from automation systems. The lowest level includes all kinds of sensors and actuators like devices from the HVAC system. The next level is for bridging and collecting resources. Embedded devices can control basic functions of the underlying level. This level is optional and can be left out. The top level – level 1 – accommodates the functions of the subsystems and transfers them into a service oriented context. The gross data rates of the bearer systems have to be adapted to the underlying subsystems. In this case a factor of 10 was chosen to allow several gateways being connected while ensuring that the capacity of the bearing system is sufficient. The needed gross data rates depend on the kind and amount of data transferred to the next level. For normal operation the full capacity of each level won't be reached.

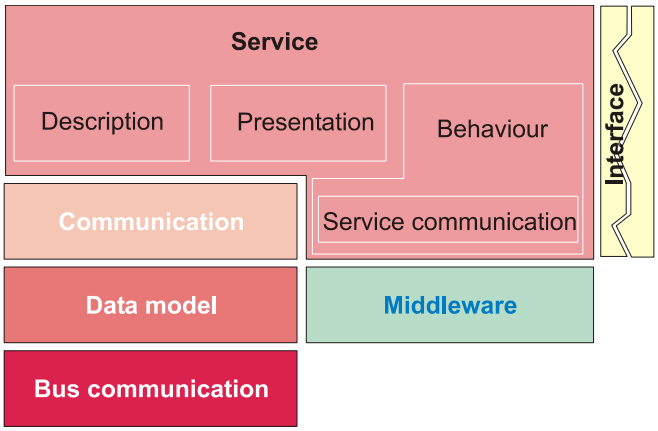

Fig. 2: Simplified gateway model

Fig. 2 shows the simplified gateway model. Using this model allows four kinds of communication paths: the bus communication on the underlying level, the communication using a simple wrapper method, a communication between the data models and a service-service user communication. This flexibility opens up a broad variety of bus systems being incorporated in this system architecture. Each gateway maps the underlying sub system to the service oriented approach and includes a description of the service/service interface and a representation of the service-device mappings. Middleware will be used to implement the system model. The middleware is the glue to map the functionality of devices to services.

By defining services the scope is moved from an implementational view to a functional view of the devices connected to each subsystem. This allows connecting devices of different subsystems. The first step was to develop a basic service which is available for every device in the system. This basic service defines an interface to get information about the device's service capabilities as well as providing a graphical user interface.

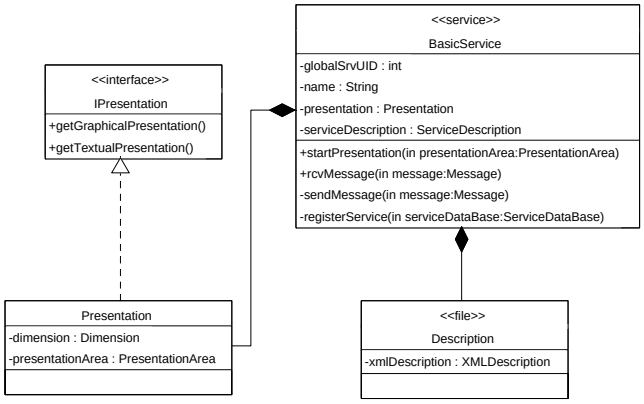

Fig. 3: Main aspects of the basic service

The figure above shows the conceptional layout of this basic service. There are elements for supplying a general description and methods to access the graphical representation of the extending service. The methods *rcvMessage* and *sendMessage* are placeholder for the service communication which is provided by the extending service. The description of the service is shown here by using an XML file. It is also possible to include a service description in a different way. The point of the service description is more abstract to make services and service users interoperable. Only by having a service description clients can use the services provided by a device. Moreover clients can use services they never knew how to handle.

An example for a more specific service is shown in fig. 4. This is already an implementation of the service. The *FileService* inherits the characteristics of the basic service and extends the functionality by adding further messages like *retrieveInformation* and *storeInformation.* The task of this service is to provide a mean for storing and retrieving any piece of information.

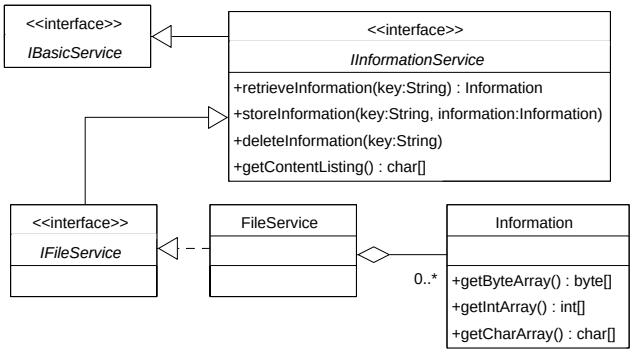

Fig. 4: Information service – general structure

3. IMPLEMENTATION

Fig. 5 gives an overview of the system as given by the research platform.

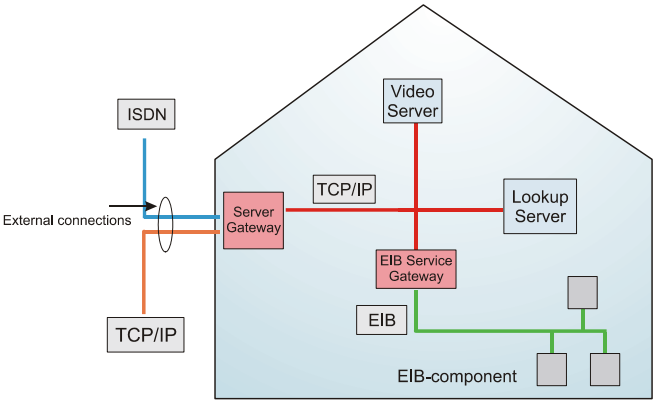

Fig. 5: Network structure

Logically the server/gateway can take care of the functionality of the EIB service gateway. Because of security concerns this approach was not chosen. The gateway is equipped with a firewall to restrict the access to the exported services.

The server/gateway has two external connections serving requests for remote control from external users: a permanent internet connection and a dial-in connection. The gateway collects all available services offered by the internal subsystems.

The infrastructure of the tele-Haus allows integrating the EIB in the system concept. To get access to the European

Installation Bus a component called TPUART is used. The TPUART is an ASIC which implements layer 1 and parts of layer 2 of the ISO/OSI protocol stack as defined by [1]. The rest of the communication stack is implemented in software of an embedded PC acting as a service gateway to the EIB. This software handles the communication with the EIB and serves all incoming telegrams and builds a process image of the actual state of the EIB devices. All data is stored in so called communication objects which bind system variable contents to system ID's. These variables can be shared all over the system (e.g. hold the status of an actuator for regulating the heating). The embedded PC keeps track of these variables to make them accessible to high layered applications. The main task of the EIB service gateway is to export all services to a high data rate network in this case TCP/IP (over Ethernet). On this level of abstraction all the EIB communication is translated in well defined services. This concept can be enhanced on all kind of bus systems. For sharing these services a middleware called Jini is used. Jini is a system from Sun Microsystems to dynamically announce, find and use different services in a network. This can be compared to Universal Plug and Play.

The Jini EIB services extend the demands of the basic service as introduced in the previous section. Each service provides a graphical user interface which can be used by means of the methods included in the *BasicService*. For demonstration purposes a simple client was developed. This client has no knowledge about the devices and does not require a static configuration. The client uses the lookup service to find all EIB devices and displays them according to additional rules dynamically.

Additional services to the ones of the EIB are video surveillance and any kind of other services which can be taken from various kinds of bus systems. There is a camera which is connected by bluetooth and also includes it's service into the network.

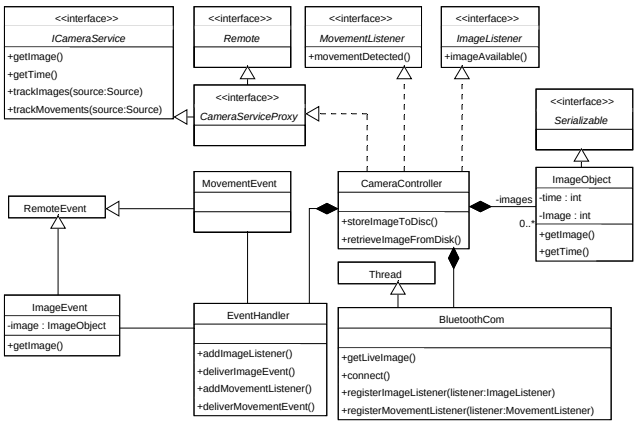

Fig. 6: Bluetooth implementation using a proxy

The service implementation is divided into parts for communication with the bluetooth camera using the Basic Imaging Profile as defined in [6]. The class *BluetoothCom* cares about requesting images from the camera. The images are stored in *ImageObjects* and can be retrieved by using a service or by calling the display service inherited by the basic service.

There are plans to integrate a Home Audio Video interoperability (HAVi) network as well. It uses a high performance serial bus (IEEE 1394) according [2] to transport data.

The basic services are defined by software interfaces in Java. Each device offering a service will announce it to the network. Using a special mechanism for discovering these services each node in the system can use another nodes service. All services can be removed and added dynamically, that means during runtime without disturbing other services. An example for the procedure to find a displayable EIB service is shown in fig. 7. The service is first registered at a central server. It can be found by the client using the lookup service. After getting the service for a device, e.g. a light switch, the client just calls the show service which causes the device to be displayed. The user can now easily control the device. The client needs no knowledge about the user interface. All needed information is transferred by the server.

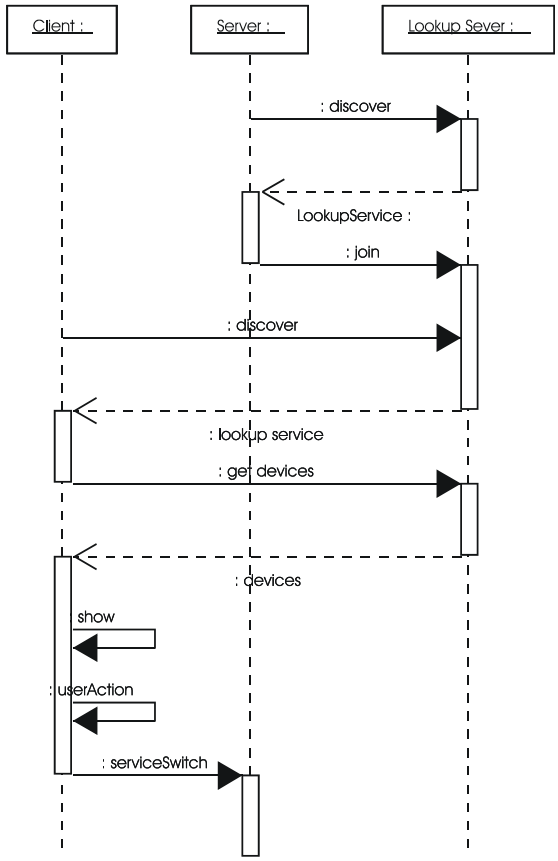

Fig. 7: Sequence diagram for finding and using a service

The service description is only needed of the client does not have any information about the usage of that service. Although the basic interface has to be well known inside the system. In the previous example the client knows that there is an interface providing methods for delivering a graphical user interface. Assuming all clients can handle Jini the usage

of services can be discriminated by the prerequisites of the client (service user):

- the client does not know anything about the service
  There are constructs in Java to get information about objects, including their implemented interfaces, parent classes, fields and methods. All these information can be gained by using reflection. With the aid of reflection it is possible to built a client which does not know anything about the service itself. Some constraints still exist: The information about the object must be loadable dynamically during runtime. Otherwise the Java Virtual Machine is not able to reconstruct the remote objects.
- the client knows the basic service
  The client can immediately access a graphical user interface to start up a service session in interaction with the user. The description delivered by the method *getDescription* inherited from the *BasicService* includes all information about the functionality of the service. The usage of some service may still require reflection to accommodate unknown data (object) types.
- The client does know the service interface
  The client is programmed to interact with a well known service. In this case the service description is not needed.

Using reflection always tends to crack up the OOP approach of Java and to fall back to procedural program schemes. Although this is the only way to make a universal service usage possible.
All the statements made above refer strictly to Jini: it does not define a basic set of interfaces which leads to vendor specific service interface implementations.
To bundle all kinds of services another piece of software exports the functionality into the Open Services Gateway Initiative (OSGi, [3]) framework. The framework defines application interfaces to install, remove, start and stop services. In this case a bundle is installed to remotely access the gateway using a java applet in an internet browser. The applet can be started at the client side and allows access to parts of the services running inside the building. The internet browser simplifies the usage of remote control software. The applet code can be downloaded from a web server and all needed (client/server) software can be updated using the OSGi framework. Another reason not to take a Jini client lies within the nature of the lookup service. The standard implementation of Jini allows two kinds of mechanisms for finding a lookup server. The first one uses UDP multicast telegrams to get information about the IP address of the lookup server, the second is unicast, but requires knowledge of the network address. UDP multicast telegrams are not routed over the internet, therefore the standard mechanism fails. The second, unicast mechanism would be suitable, but does not fit into the Jini concept not needing to know where the lookup server is located.
The communication with the gateway is secured by using Secure Sockets Layer [4]/Transport Layer Security[5]. This allows mutual authentication by certificates.

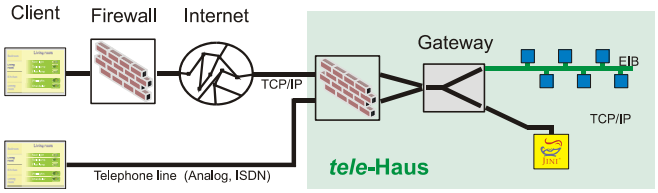

Fig. 8: Firewall configurations

There are two cases having to be reviewed when connecting remotely to the OSGi gateway: in the first case an additional firewall regulates all incoming and outgoing connections from and to the client. In the worst case only a few TCP/IP ports are left open for outgoing communication and all incoming connection requests are blocked. The implementation of the remote control software allows even to pass this configuration by initializing an outgoing connection to the server and using it also for data exchange initiated by the server. Using a dial in server (fig. 8, bottom) these restrictions do not apply.

The developed system concept includes several aspects for connecting many bus systems and make them remotely available. This allows parts of the available services, which can be found inside the internal network, to be exported to an external user. An example is a device proxy for a EIB consumption meter to be exported to the OSGi gateway for enabling remote readings from the energy supplier. Besides using this technique for remote energy readings it can be used to detect system failures remotely enabling facility management companies cost effective analyzing tools not only for private homes.

4. CONCLUSION

The concept described in this paper is using standard technologies for communication between devices. The focus is not only set on making devices communicating with each other. This could be achieved by proprietary gateways as well. Furthermore an abstract interface is defined which enables devices from several bus systems exchanging object data. This data can consist of a single command for switching lights but also a video stream from an IEEE1394-1995 network or a TCP/IP video stream. The protocols at very low layers are hidden and not of importance anymore. This unifies the access to services and allows the user to take advantage of all installed bus systems and enhance their possibilities by linking them together.
Although Jini is meant for distributed systems it is not suited for being exported to the internet. The used UDP-telegrams for announcements are not routed through the internet. This is why OSGi was taken to export the services. The gateway can be extended by adding further bundles for each bus system installed in the building. This combination of in-house integration and export of services makes the system highly extensible without demanding for proprietary software for the benefits acquired by coupling several systems together.

REFERENCES

[1] EIBA, “The EIB Handbook 3.0”, Brussels, 1998.
[2] IEEE, ”IEEE1394-1995, IEEE Standard for a High Performance Serial Bus”, 1996.
[3] OSGi, OSGi Service Platform, ISBN: 1 58603 252 6, 2002.
[4] Alan O. Freier, Philip Karlton, Paul C. Kocher, “The SSL Protocol Version 3.0 Internet Draft”, Netscape, March 1996.
[5] C. Allen and T. Dierks, “RFC2246: The TLS Protocol Version 1.0”, January 1999.
[6] Arai, Tatsuo; YAMAMOTO, Ryohei; RANG, Maria; et al. : Basic Imaging Profile Interoperability Specification – Version 0.95c. 2001.
[7] Halsall, Fred: Data Communications, Computer Networks and Open Systems. 4. Ausgabe, Harlow : Addison-Wesley, 1996.
[8] Werthschulte, Kay: Integration von heterogenen Bussystemen in die Heimautomatisierung unter Verwendung von Middleware. Dissertation, Technische Universität München, 2003.

Author: Dipl.-Ing. Kay Werthschulte,
Prof. Dr.-Ing. Friedrich Schneider,
Technische Universität München
Institute for Measurement Systems and Sensor Technology
D-80290 Munich
Germany
Tel. +49 89 289 23350, Fax. +49 89 289 23348,
k.werthschulte@ei.tum.de.