CANeye - controlling and monitoring CAN via web interface

Jean Randhahn, Helmut Beikirch – Universität Rostock

In the last years Ethernet has found widespread use and acceptance in office communications. Today it is readily available in almost every single personal computer for very low cost. With the growth of Ethernet based networks within offices the desire to connect to the field bus dominated production floor has grown as well. There are many different motivations for this desire, e.g. to monitor production flow and machines over short and medium distances or to enable the use of readily available hard- and software to name a few.
The following paper will describe CANeye, a low cost embedded web server that is able to connect to CAN. CANeyes position and abilities within the network and an introduction to its hard- and software structure will be covered.

I. Introduction

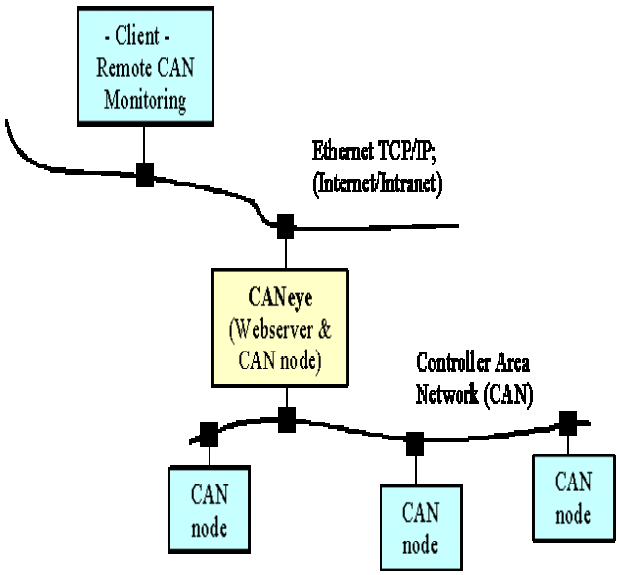
The goal was to develop a device that allows access to a field bus from Ethernet TCP/IP networks. CAN as field bus had been chosen since our institute has a special focus on CAN but also for the possible applications in CAN dominated areas. Later ones include multimedia and motor service applications for automobiles and medical applications that need unobtrusive monitoring of bus activity.
Thus CANeye, a low cost embedded web server, has been developed. It contains a full function active CAN node and an Ethernet-TCP/IP interface that enables communication with the implemented HTTP server. This CANeye is able to send and accept CAN data dynamically as html-, text- or raw data file. To acknowledge demands on power consumption, size and price a microcontroller from TI's MSP430 series is being used.

II. CANeye within the network

CANeye is meant as a connecting device between Ethernet and CAN. Within the networks it takes a similar position as routers and gateways do. It includes a 10BaseT (10 Mbit/s) Ethernet interface that connects to standard office networks via router, switch, hub or directly to the client. For CAN the interface can be a simple two wire connection or a combined data power connection. For a connector a DSUB9 is being used.

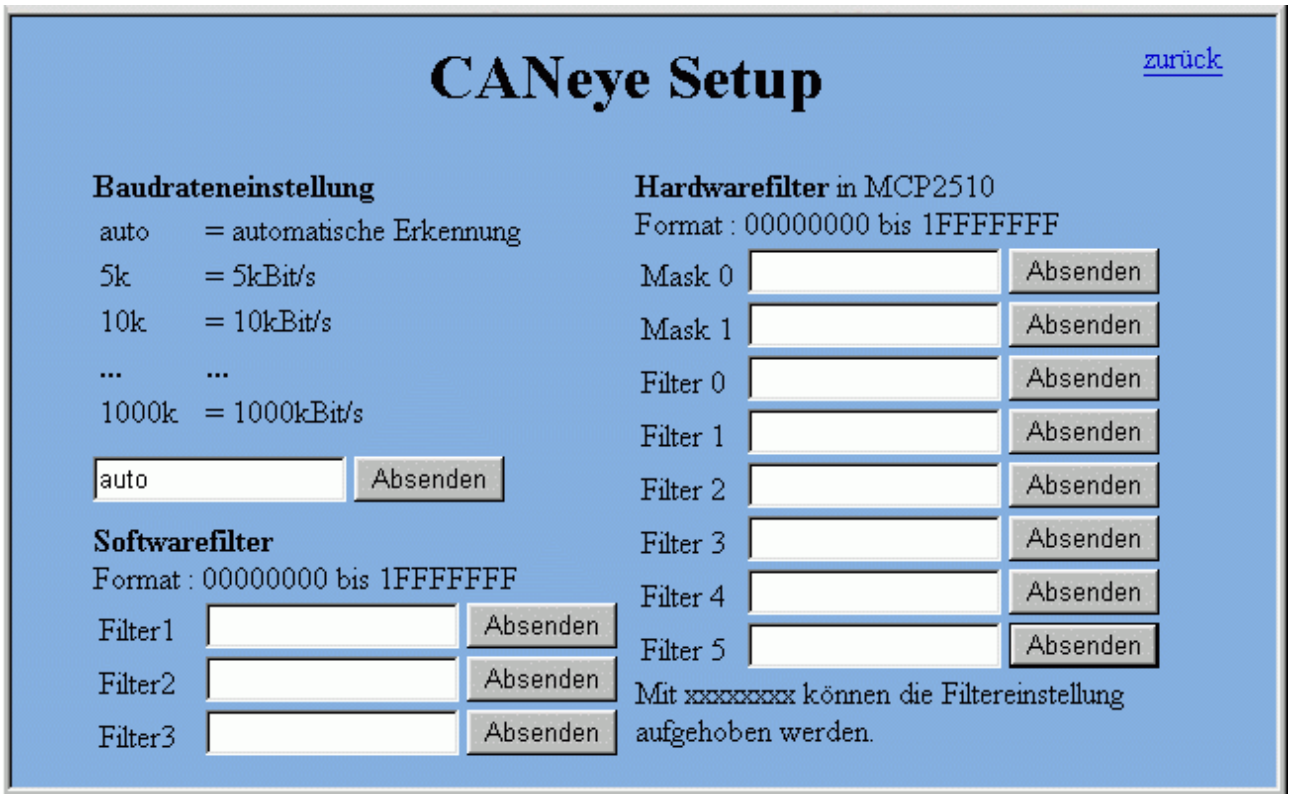
III. Functionality

CANeye serves two main purposes, which are receiving data from CAN and sending data to CAN. Both actions are executed upon request by a client. Depending on the application, however, both actions are not always available. Their availability depends whether CANeye is



Picture 1 CANeye in sample network configuration

run in active or passive mode.
The passive mode will not allow the client to send data to the CAN. Also no data packet that is received by CANeye from CAN is being acknowledged. Effectively this allows the client to unobtrusively monitor the CAN. This can be critical in safety sensitive applications.
The active mode will allow receiving as well as sending data to CAN. In this mode CANeye works as a common active node on CAN. This mode is suitable when interacting with devices that are located on the CAN is necessary.
Data that is received from CAN can be filtered to collect only the data the user needs. This is done by setting up the filters in the CAN controller or by setting software filters in CANeyes software. Both can be set by the user via html pages that are accessed like pages from any other web server.
The received data is made available in different ways. When used in conjunction with



Picture 2 *setup page for CAN parameters*

a standard personal computer that has a internet browser available, the data can be viewed in a html document that is dynamically generated. For further text processing the data can be viewed as text file as well, by same means as the html file. In both cases the data can be positioned anywhere within the html or text document. This enables the user to prepare a form and have the data being filled in dynamically upon request.

For more effective further processing of received data it can be made available as raw data. The data will then include all known details like length of frame, identifier, frame type etc.

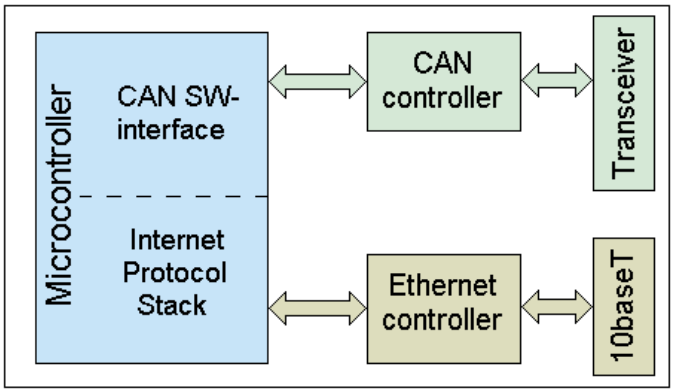
Data that CANeye is to send to CAN can be sent through a html form. This allows the use of the send command within CANeyes web page as well as from other HTTP based programs. The data needs to contain identifier and data at the minimum, all other information for the CAN data frame can be filled in by CANeye with default values.

Administrating CANeye is easy by using the built in configuration web pages. They allow changing the CAN baud rate that is detected automaticly on startup or upon request, as well as setting the earlier mentioned filters. TCP/IP parameters like IP-Address and Subnet can be changed via the configuration pages as well. When using CANeye manually only infrequently, the user can read the online help manual, that is stored in CANeye.

IV. Hardware

CANeyes hardware consists of three controllers. For communication it uses a standard Ethernet device, a CS8900A from Crystal Inc., and a stand alone CAN controller, a MCP2510 from Microchip Inc.. Since the Ethernet controller integrates a physical interface only a signal transformer for isolation and a handful of passive components are needed to enable the Ethernet interface. The physical interface is a 10BaseT, which uses twisted pair cable at 10 Mbit/s. To allow time for processing frames by a microcontroller between two arriving data frames, the Ethernet controller has an integrated 4 kB RAM to store unprocessed frames as well as frames scheduled for sending. The microcontroller interface is a 16 bit wide bus, that can be used in 8 bit mode, too. CANeye implements the 16 bit bus and can make use of the receive interrupts.

The stand alone CAN controller implements full CAN 2.0 A/B at 1 MBit/s. It supports active and passive mode, which allows its usage for unobtrusive message reception. The controller includes two receive and three transmit buffers and is capable of message filtering. For connection to the CAN it needs a standard CAN transceiver. For interfacing to a microcontroller it contains a Serial Peripheral

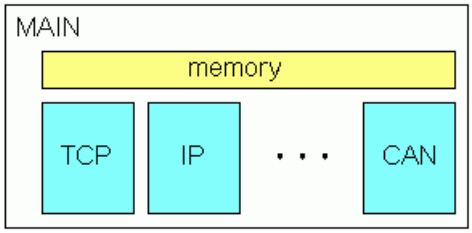


Picture 3 *illustrates the simplified schematic*

Interface (SPI).
A microcontroller, a MSP430F149 from Texas Instruments Inc., is used as main processing device. It is a 16 bit MCU that requires only very little power. It includes a SPI to interface to the CAN controller used and enough I/O pins to interface to the Ethernet controller in 16 bit I/O mode. A RAM area of 2 kB and FLASH program memory of 60 kB are at the very low end of internet enabled devices but sufficient for the task.
The low power requirement is another distinctive feature of CANeye. All components used run at 3.3 V and when all components are active the whole device requires 50 mA at the most. Because of the low supply voltage the device can be run from a common 5 V wall plug or it can be powered by a CAN cable that includes 5 V supply lines. This option can be manually jumpered and keeps external wiring to a minimum.
The printed circuit board (pcb) that holds all components and connectors is a simple 2 layer pcb with a size of approximately 3“ x 2.5“. The complete schematic is available at randhahn.de for free download and use.

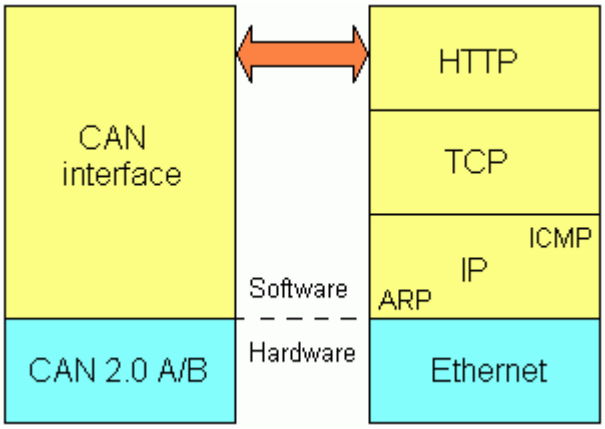
V. Software

Basic concept - The ISO/OSI model states that communication between computers can be defined in various layers that allow to distinct certain features of the communication process. These layers are often represented by one sometimes multiple communication protocols. In CANeye this concept has been employed to an extend that every used protocol is implemented seperatly. Furthermore are all protocols realized as a finite state machine. Finite state machines, Mealy state machines in this case, are inherently robust and allow for easy change and extension of functionality. They also help to make the software easier to read and document.
The protocols and thus the state machines need to process and pass on the received data frame to the next protocol. This is handled by a simple main task that provides a mailbox like communication system. Whenever a data frame is received it is dropped into a free mailbox, given a destination and signed by the sending protocol. The main task also provides stubs for running one or more state machines in cycles. During one cycle all state machines are activated at least once. Upon activation the state machine checks the mailboxes, processes the data frame and quits. One advantage of this method is, that the little memory that is available, can be used by all state machines and be used in a very predefined way. There is also a lot of room for further enhancements like reseting a single state machine upon failure, which is possible since all state machines are independent of one another and only share a common set of commands and data structures, but not actually memory space or functions.
The use of state machines and the way the memory is provided to them makes the whole system very predictable. In time the software acts asynchronously, since the activation of a single state machine in a single cycle cannot be predefined and depends on pending events.
Connecting to CAN - Almost all necessary layers for CAN are implemented by the CAN controller. There is no higher protocol like CANopen implemented as it is not needed for CANeyes functionality. All that needs to be done is to transfer frames to and from the CAN controller. Also a set of functions for initializing and changing parameters is needed. All these are put into a single state machine. It always checks for new frames on the CAN controller and for messages in the mailbox.
Upon Initialization the CAN controller is put into passive mode and the currently active baud



Picture 4 basic software concept

rate is obtained by changing the baud rate until no bus errors are encountered any more. This is a simple and fairly fast way to determine an unknown baud rate. A drawback of this method is that it presumes activity on the CAN that has to be free of errors. The active mode can be set by a switch on the configuration web pages. Also the masks and filters can be set this way as was earlier mentioned. Their initial state will allow all frames to pass. The filter scheme provided by the CAN controller used enables the detection of single identifiers as well as identifiers that fall through a scheme. They are referred to as filter and mask. In a separate memory location provided by main the frames are stored according to their identifier.



Picture 5 structure of communication stacks

Connecting to Ethernet-TCP/IP - To be able to communicate with a client via TCP/IP quite a few layers and thus protocols need to be implemented. A basic stack of protocols has to contain means to address nodes, to resolve these addresses, to open a connection to them and to transport data. Even a basic protocol stack allows for quite some flexibility and different scenarios that are not needed for creating a server in general and CANeye in particular. For this reason only required functionality is implemented for most protocols in CANeye. However, CANeyes protocol stack is conform with the internet requirements.

Ethernet is the medium used for interconnecting clients and servers. It provides basic data frame transport and little means to ensure transport safety, like retransmission and collision detection. Its carrier transmission access protocol is CSMA/CD, which is similar to what is used by CAN, though it does not avoid collision but only detects them. Ethernet is almost completely implemented in hardware. Even though it provides a mean of addressing, it is not used by the internet stack as this addressing is hardware dependent. There is a IEEE standard for Ethernet, but since the original draft was written and maintained by a consortium there are now two different protocol versions available. The widest acceptance can be claimed by the improved original draft Ethernet V2. The IEEE standard is often referenced by device manufacturers but not fully implemented. CANeye works with Ethernet V2.

The Internet Protocol (IP, version 4) supplies a mean of addressing to the internet stack. The addresses are hardware independent and are split into categories. These categories are meant for networks of different sizes and for multicasting. To split these categories up into even smaller nets subnet masks are being used. They allow for a more efficient use of the address range that is available. CANeyes address will be set upon programming the device. This address is not static and can be changed to fit needs via the configuration web pages later on. The same holds true for subnet masks used.

Another feature of IP is its capability of fragmenting datagrams that are too large for the transport medium. This fragmentation mechanism splits the datagram into smaller chunks, sends them off and reassembles them in correct order at the receivers end. CANeye does not support fragmentation, as it has too little memory available for such a task. This does not necessarily mean that client request with too large datagrams cannot be handled. Keeping the datagrams to a predefined maximum size can be handled by higher layer protocols and will be shows later on.

The Address Resolution Protocol (ARP) enables the internet stack to resolve IP addresses. This is done by a broadcast on Ethernet. Since CANeye is only meant as server it does not need to resolve addresses itself. All requests will already contain the senders address. All CANeye has to support are ARP requests towards itself, which is simply to generate a reply. To remember addresses used during a current connections a table of addresses has to be kept. Addresses

in this ARP table need to be cleared on a periodic basis to allow for address changes of clients because of reboots or similar events. This mechanism is fully implemented.
The Internet Control Message Protocol (ICMP) is a supporting protocol to IP. This protocol is used to exchange error and network information. It is not necessarily part of a basic stack, but provides some very useful tools for network monitoring, like ping. Ping is simply a datagram that is sent to an internet node and has to be answered with a copy of that datagram. Though simple it allows to gather much information about the state of the internet node and the network. Ping is implemented in CANeye.
The Transmission Control Protocol (TCP) provides all services that have not yet been covered by the lower protocols. These are mainly connection establishing, transport safety, flow control and multiplexing.
Connection establishment is handled via a three way handshake and will result in a point-to-point connection between two processes of two network nodes. Transport safety for the following data exchange is guaranteed by error checking, end to end control and flow control. Error checking is provided by a checksum, that is generated from the TCP header and the data within the TCP datagram. Encountering an error causes a retransmission of the data by simply not acknowledging the reception of the erroneous datagram. End to end control is realized by acknowledging all data received. Numbering all datagrams to ensure the right order on the receiving node and mechanisms to prevent receive buffers from overflowing on both ends provide good means of flow control. All these so far mentioned functions are implemented in CANeye and necessary for correct operation of the internet stack. The mechanisms to prevent buffer overflow are used by CANeye to keep the datagram size to a value that can be handled with the little RAM available. There is the maximum segment size that is agreed upon connection establishment. It is used to keep single datagrams small enough for CANeye. The so called window mechanism that is directly responsible from keeping buffers from overflowing is used to restrain clients from sending more than two datagrams at a time. Thus CANeye can handle with only little memory all data traffic and still stay within the internet requirements.
Another service TCP provides is multiplexing connections. That is when multiple clients want to communicate with the same process on one computer, e.g. a web server. For this purpose TCP creates sockets that contain information about the clients connection info and the server connection info. Obviously the memory demand of these sockets grows rapidly with increasing connection request. For this reason CANeye supports only one TCP connection at a time. Theoretically this will limit data transfer significantly. Real life experience has shown that connection request occur seldom at intervals that are close enough to make multiple connections necessary. Thus even though only one connection can be handled multiple requests can mostly be served without delay.
On top of this internet stack the web server is located. It uses the common Hypertext Transfer Protocol (HTTP) to communicate to clients via the TCP/IP connection. In CANeyes case the web server has to be able to serve static as well as dynamic files. Furthermore it has to be able to handle action requests. HTTP version 1.0 provides a sufficient set of commands for this purpose. HTTP/1.0 is currently the most widespread HTTP version and is compatible to the older version 0.9. HTTP/1.0 also provides enough error codes to show the client which actions are possible and allowed and which not. HTTP is commonly used as basis for internet browsers, but can also be used by other programs. Because of this CANeye can be used in conjunction with a browser or with other programs.
Web pages that are served by CANeye reside in its program memory. These pages can be static and served as is or they can be dynamic. Dynamics are created by providing a mean to insert variable data into the page while serving it. For this purpose Server Side Includes (SSI) are used. SSI defines a standard of marking the places to insert and the name of the variable that has to be inserted. There are many predefined variables like server IP, date, time, but also user definable variables. These are used to include the CAN frame information into the web page. For each CAN frame saved a seperat SSI variable for length, identifier, data etc. exists. This way CAN frames can be inserted and displayed according to their meaning by proper design of the web page. In this way HTML- and text-files can be served. To create a raw data file the same mechanism can be used, but the file has to be prepared differently. By leaving the file blank except for the proper SSI commands it is possible to

serve raw data files.
While file requests are commonly invoked with HTTPs GET command, action requests use the POST command. The POST command itself is invoked by using a form in a web page or directly by the clients program. The actions are identified by parameters in the POST command, these also contain values when the action requires one. These actions are used for setting parameters of the internet stack and CAN as well as for sending frames to CAN.
Since TCP supports only one connection at a time the web server does so as well. This greatly simplifies the server design.
Following is a list of important supported and unsupported features :
CAN
- passive and active mode
- automatic bit rate detection
- supports basic and extended identifier
Ethernet
- packet format: Ethernet V2
- broadcast and individual addressing
IP
- version: IPv4
- no options
- no fragmentation
- broadcast, incl. in subnet
ARP
- reduced to reply
- dynamic address table
ICMP
- ping supported
TCP
- only one connection supported
- passive open/close, active close implemented
- retransmission time out in connection states
- give up time out at establishing connection
- delayed ACK during data transfer
- maximum segment size option supported
- window mechanism implemented
HTTP
- version: HTTP/1.0
- GET and POST supported
- static and dynamic file serving
- one connection at a time supported

VI. Results

CANeye is a device to make CAN data frames available on TCP/IP networks as they are used in office environments. It is a stand alone device that requires no special software, but a commonly available internet browser. Even online help or simple documentation can be stored on it and thus made available at the time and place it is needed. By using already available infrastructures and software it can be integrated into existing structures easily and cost effective. Being able to connect a field bus to the office network its largest potential lies in closing the gap between production floor and office. Because of it small size and power requirements it can also be used as a field device for maintainance and monitoring tasks.
Also CANeye proves that it is possible to enable internet communication on resource limited platforms. It furthermore shows that no unacceptable deviations from the internet requirements are necessary to accomplish this task.
All schematics and more info are available for free download and use at http://caneye.de.

Recommended reading and sources:

[1] Stevens, W. Richard: „TCP/IP Illustrated, Volume 1: the protocols.“ Reading, Mass.: Addison-Wesley Pub. Company, 1994.

[2] Hein, Mathias: „TCP/IP – Internet-Protokolle im professionellen Einsatz – 5., akt. und erweiterte Aufl.“, Bonn: MITP-Verlag, 2000.

[3] Randhahn, Jean, „CANeye - Entwurf und Implementierung eines aufwandsminimierten embedded Web Servers in einem CAN Knoten“, Diplomarbeit, FH-Stralsund

[4] Lawrenz, W.: „CAN Controller Area Network“, Heidelberg: Hüthig Verlag, 1999

Dipl.-Ing.(FH) Jean Randhahn
Universität Rostock
Graduiertenkolleg „Integrierte fluidische Sensor-Aktor-Systeme“
Albert-Einstein-Str. 2
D-18051 Rostock
Tel: +49(0)381 498 3516
Fax: +49(0)381 498 3608
Email: jean@randhahn.de
Web: http://randhahn.de

Prof. Dr.-Ing. habil. Helmut Beikirch
Universität Rostock
Fachbereich Elektrotechnik
Institut Gerätesysteme und Schaltungstechnik
Albert-Einstein-Str. 2
D-18051 Rostock
Tel: +49(0)381 498
Fax: +49(0)381 498 3608
Email:helmut.beikirch@etechnik.uni-rostock.de