Design of an enhanced TP-UART based KNX PC interface

Georg Neugschwandtner Andreas Fernbach

Automation Systems Group
Institute of Automation
Vienna University of Technology

Treitlstraße 1-3, A-1040 Vienna, Austria
{gn, afernbach} @ auto.tuwien.ac.at

The TP-UART-IC provides a convenient basis for a versatile KNX PC interface. However, its host protocol contains timing constraints that are difficult to meet on a PC. This paper presents the design of such an interface that addresses these issues without sacrificing flexibility. It also documents properties of the TP-UART host protocol that were uncovered during development and are not described in its data sheet.

1 Introduction

Although designed for interfacing microcontrollers that operate KNX devices to the TP1 medium, the Siemens TP-UART-IC [1] – in the following simply referred to as "TP-UART" – can also be used to build a simple yet highly versatile TP1 network interface for PCs. For example, such an interface has benefits in a lab environment since the TP-UART host protocol places very few restrictions on the frame format. While the TP-UART handles most of the KNX protocol stack up to the data link layer (and thus, the most critical timing requirements), higher level protocol aspects remain with the host controller. This leaves ample leeway for, e.g., testing protocol extensions.

A TP-UART based PC/KNX interface may also have benefits in the early prototyping stage of (TP-UART based) embedded KNX devices. In addition, it can be a simple solution for traffic intensive applications (e.g., visualization software) that require an interface with wirespeed performance. Therefore, it is not surprising that various TP-UART based PC/KNX interfaces have seen the light of day.

Besides the TP-UART IC itself, such an interface must at least include the support circuitry required by the TP-UART and convert between standard logic levels and those used by EIA/TIA-232 (in the following referred to as RS-232 for brevity). A design that also provides galvanic

separation between the KNX network and the PC was used and published in 2001 by the FH Deggendorf [3] and was converted from surface-mount to pin-through-hole components at the Automation Systems Group, TU Vienna in 2003 [3]. Disch Systems offers a product that adds an enclosure as well as access to the temperature warning and protocol speed signals of the TP-UART [5]. The Siemens “Bus Transceiver Module” [6] can also be used if combined with a KNX serial interface application module. It then provides access to the TP-UART reset and link power failure warning signals.

However, the TP-UART host protocol – being designed for microcontroller interfacing – still contains some relatively tight timing constraints. While these are not a serious problem for a microcontroller to handle and can be considered a fair price to pay for the flexibility of the protocol, a PC without a real-time operating system will hardly be able to meet them.

Linux kernel drivers have been developed to address this problem [8, 9, 11]. However, a kernel driver is operating system dependent and requires significant maintenance effort (cf. [10]). Moreover, it introduces an additional proprietary protocol layer, which can be an obstacle if program code is to be transferred to an environemnt where such a driver is not necessary. Operating system independent user mode solutions are possible [12], but require assumptions about the frame format and special attention to recovery from error conditions.

The goal of the project described in this paper was to obtain a solution that would allow to use the full flexibility of the TP-UART host protocol through a generic, operating system and programming language neutral serial port abstraction. Any modifications to the protocol were to be stateless and as small as possible.

This was achieved by placing an MCU (microcontroller unit) between TP-UART and PC which monitors the timing of the critical TP-UART services and converts it to out-of-band communication where applicable. An appropriate hardware platform was designed that allows putting this approach into practice conveniently, but is versatile enough not to be limited to this single task.

In the following, the interface hardware and how it can be used to address the TP-UART host protocol issues are described. Dealing with these issues requires detailed information about the protocol. Thus, tests were made to complement the information given in the data sheet [1]. Results obtained in the course of these efforts are described as well.

2 Hardware

The design of the interface hardware follows the overall goals of being simple to build and easy to handle. Also, no particularly expensive software or hardware tools should be required.

Still, the interface should provide a certain amount of versatility. In-system programming and debugging is standard on current MCU types and was to be included. However, there was also to be a minimal user interface to control the MCU behaviour as well as to trigger actions (such as sending a test network message) and show status information (e.g., error conditions) without reprogramming, requiring a debug interface connection or using the RS-232 interface for this purpose. Also, access to the TP-UART status signals should be possible.

The KNX TP1 medium is electrically isolated to ground. On the other hand, the RS-232 interface on the PC is grounded. Therefore, galvanic isolation between the KNX and the PC side of the interface should be provided. Although the interface is primarily intended for laboratory

use, this was made a requirement to avoid problems in case the interface should be connected to a larger KNX installation.

2.1 Selecting the MCU

The MCU is a key design component. Within this project, its task is not resource intensive with regard to processing power, memory size or I/O. It mainly consists of communicating via two relatively low speed UARTs (Universal Asynchronous Receiver Transmitter) and checking the timing of incoming messages. Thus, it can be optimized for low power consumption and easy manual soldering. In line with the overall project goals, the price of the programming adapter and toolchain necessary for developing and debugging the MCU software also have considerable impact.

First, MCU types available in packages with low pin count and large pin spacing (at least 1 mm) were selected from popular product families. Evaluating the types on this shortlist according to the above criteria, the TI (Texas Instruments) MSP430-F123 [14] (and its F1232 sibling [15], which can be considered identical for the purposes of this project) emerged as ideally suited for the project.

It provides a 16 bit RISC CPU, 8 kB Flash memory, 256 Bytes RAM, and can operate at clock rates of up to 8 MHz. Even at 8 MHz, it consumes less than 3 mA in active mode. In-system programming and debugging are possible via a standard JTAG (Joint Test Action Group) interface. Low-cost JTAG adapters are available. Entry level versions of two commercial toolchains are offered free of charge. In addition, a GCC based open source toolchain exists. The MSP430-F123 is available in a 28-pin small-outline package with 1.27 mm pin spacing (the same as the TP-UART-IC has).

The MSP430-F123 has only one hardware UART (no MCUs in this class appear to be available that would have two; rather, many have none at all). This means that the UART function has to be implemented in software, using a hardware timer. While the MSP430-F123 has only one 16 bit hardware timer, it is equipped with three capture/compare units. Using one of these units for software UART transmission and one for reception, full duplex operation with minimum CPU load is possible. The third capture/compare unit remains free for application timing tasks. TI even provides a software UART library, which, although it did not satisfy all requirements (no parity support, insufficient performance), could be used as a starting point.

2.2 Power considerations

On a cluttered lab desk, any wire that is not strictly necessary is a nuisance. Therefore, the interface should not require an external power supply. Thus, the required power must be found elsewhere. Actually, since the interface is to provide galvanic separation, even two separate power sources are needed.

The TP-UART provides a stable 5 V supply derived from the KNX TP1 network out of its VCC pin. However, according to its data sheet, the maximum load must not exceed 10 mA. This is used to power the MCU, the user interface, one half of the optocouplers, and any supporting circuitry. Since the MSP430 requires a 3.3 V supply, a voltage regulator is required, whose ground pin current also factors into the equation. However, regulators with a ground pin current as low as 0.15 mA at 10 mA load current are readily available. Thus, if LEDs are used for

displaying information to the user, the relevant loads besides the MCU are the optocouplers (two transmitters and two receivers), which consume about 2 mA together, and the LEDs. The LEDs require 0.3 mA for reasonable brightness. If four LEDs are used, the maximum supply current available for the MCU is about 6 mA.

On the PC side, the interface needs to generate RS-232 signal levels. An RS-232 receiver considers a line voltage of 3 to 15 V ("space state") as a logic '0' and the corresponding negative voltage range ("mark state") as a logic '1'. Transmitters are required to output at least $\pm$5 V. To further increase the noise margin, levels around $\pm$10 V are typical.

Since most equipment today operates on +5 V or lower, ICs that generate these voltages from a single +5 V supply and perform the necessary signal level conversion are popular. However, there is no such thing as a +5 V power supply pin on an RS-232 interface. Power can only be drawn from the signal lines. The exact amount depends on the driver circuits in the PC. Short circuit currents of 10 mA are typical (with a single output shorted). The output voltage rapidly decreases with increased load.

In theory, it should be possible to pass the DTR signal through a voltage regulator to obtain a stable 5 V supply that is independent of the voltage the line is actually driven to by the PC, and use a standard RS-232 transceiver with built-in DC-DC converter for level conversion. In case DTR is not connected (which may be the case with some embedded PCs), this approach allows simply connecting a stock external power supply before the voltage regulator. It is also elegant since it respects the semantics of the DTR signal, which is to be asserted (in space state) whenever the PC opens the port for communication.

However, this method is not very efficient. In our experiments, we were unable to obtain stable operating conditions using DTR alone, despite using low power ICs. The optocouplers could not be further optimized for power consumption without increasing the signal rise time to unacceptable levels. Therefore, we decided to use RTS as well as a negative power supply (requiring the signal to be deasserted). This way, a (low power) RS-232 transmitter can directly be provided with positive and negative input voltages. DC-DC conversion is eliminated entirely. Correct DTR and RTS polarity is ensured via diodes and visually confirmed using a LED. If the PC does not provide an RTS output or if the signal has to remain available for communication, a $\pm$9..12 V external power supply can still be connected.

It would also be possible to derive power from signal lines irrespective of their polarity, leaving communication entirely undisturbed (cf. [13]). This possibility was not pursued since it would significantly increase circuit complexity. Rather, the goal was a simple design that would eliminate the need for an external power supply in the majority of use cases.

2.3 Design overview

The block diagram in Figure 1 shows an overview of the main parts of the TP-UART interface board. The MSP430 MCU is looped into the serial data connection between TP-UART and PC, with its hardware UART to the PC side. The MSP430 is clocked by an external 8 MHz oscillator. For in-system programming and debugging, its JTAG interface is lead onto a pin header that follows the pinout of TI's "Flash Emulation Tool" JTAG adapters. As a basic local user interface, four LEDs, two DIP switches and two push buttons are connected to general purpose I/O pins of the MSP430.

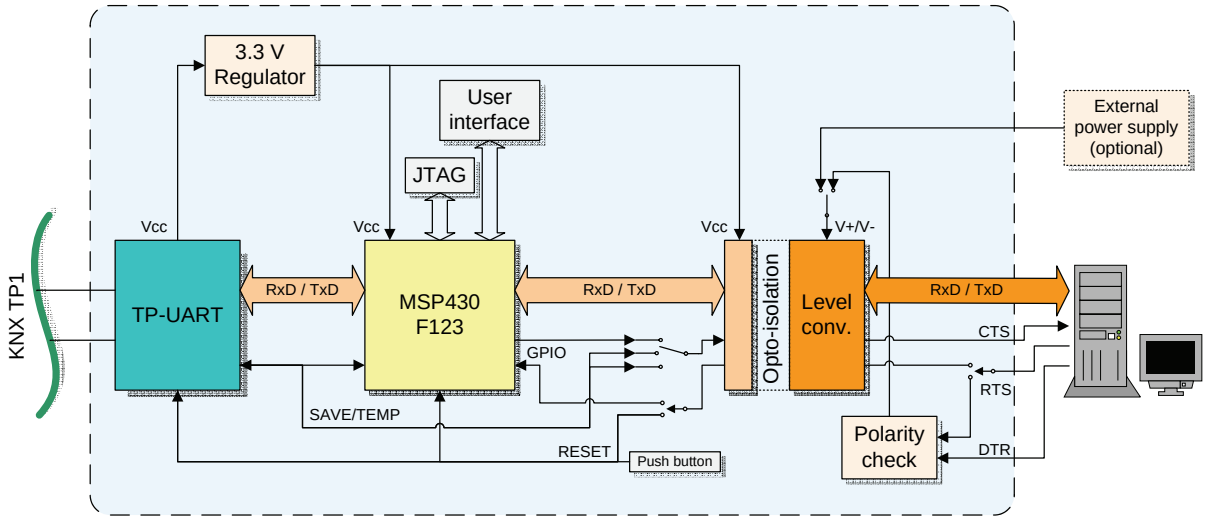


Figure 1: TP-UART interface board connected to KNX TP1 and a PC

The RS-232 serial interface to the PC is terminated by a standard DB-9 female connector. Optocouplers ensure galvanic separation and also handle level conversion for incoming signals. Outbound RS-232 signals go through a low power driver IC (e.g., Maxim MAX1488E). Two signal paths are provided in each direction. They are connected to TXD, RXD, RTS, and CTS.

Users have the choice of passing the TP-UART power loss signal (SAVE), its temperature warning output (TEMP), or a MCU general purpose digital output into the PC. The TP-UART outputs are adapted to the optocoupler input characteristics via transistors as on the TP-UART "header board" [7] (designed by Fritz Praus for internal use at the group).

If an external power supply is connected, RTS can be used to trigger the MCU and TP-UART reset inputs or as an additional input signal for the MCU. The power supply (earth free, stabilized, $\pm$9..15 V) is connected in parallel to DTR and RTS via a 3-pin header. No damage to the PC is expected in case of voltage mismatch between the external power supply and DTR or RTS, as RS-232 line drivers are required to withstand indefinite short-circuit conditions between their output and any other signal conductor.

All status pins of the TP-UART are also permanently connected to the MCU, whose firmware can pass their state to the PC via UART communication (this possibility is currently not used). For better monitoring with an oscilloscope or logic analyzer, all TP-UART data and status I/Os (TSTOUT, TxD, RxD, RESn, SAVE) are broken out on a pin header as well.

Since the TP-UART and the MCU are powered from the KNX network, they are automatically reset when link power returns after a failure or after being disconnected. In addition, a push button is provided that allows to manually reset both TP-UART and MCU if necessary. Due to its integrated brownout detection, the MSP430 reliably starts up once its power supply is stable. Therefore, a diode protects it from being reset when the TP-UART activates its RESn signal. This increases the time available for firmware initialization, ensuring that the MCU can easily catch the Reset.indication service, which is sent by the TP-UART almost simultaneously with releasing RESn. However, the MSP430 starts up fast enough to correctly receive the service even if the TP-UART is allowed to delay its startup by directly connecting the reset pins.

The support circuitry for the TP-UART-IC follows the “typical application circuit” shown in its data sheet. A jumper block determines whether the TP-UART operates in normal mode or in analog mode and connects the required passive components. In normal mode, its host interface is always running at 19200 bps. The MSP430 pins connected to the TP-UART TXD and RXD signals can be directly controlled and used as inputs by the capture/compare-units of the MCU. Thus, accurate timing and efficient operation are possible, both with the TP-UART-IC in normal as well as analog mode.

The PCB (printed circuit board) measures 80 x 100 mm. Trace width and spacing are chosen to allow low-cost production. Pin-through-hole components were used wherever possible. The layout also includes mounting holes. However, an enclosure is not required, since all components can be directly mounted on the PCB. All terminals, jumpers, switches, push buttons and LEDs are also labelled there (within the limitations of available space).

If the enhanced functionality provided by the microcontroller is not needed, the board can also be assembled without the MCU. This allows immediate use with existing TP-UART related PC software. The PCB layout includes the necessary optional pass-through tracks required for the RXD and TXD lines. The KNX TP1 connector features a dual footprint that allows either headers for standard bus terminals or spring loaded terminals to be mounted.

3 Addressing TP-UART host protocol issues

For PC software, keeping time of exactly when a character was received or sent over a serial port is not easy. Often, the available abstractions only offer access to the data stream, without any timing information at all. However, in the TP-UART host protocol, some information is available only by observing the timing of UART character transmissions.

The most critical issue concerns end-of-packet (EOP) recognition for frames propagated from the TP1 network to the host. The end of such a received frame is indicated by the TP-UART as a certain (minimum) period of silence on the host interface only.

While receiving such a frame, the host must send a U_AckInformation service to the TP-UART to indicate if the frame is to be answered by a Layer 2 Acknowledgment frame. While the TP-UART data sheet specifies a deadline for this service, it does not say what will happen if this deadline is not met.

3.1 End of packet indication

The challenge is best described by quoting the TP-UART data sheet: “The host controller has to detect an end of packet timeout by supervising the EOP gap of 2 to 2.5 ms” ([1], p. 19).

A PC application sitting atop a standard operating system and runtime library has no way of timing external events with such high resolution. An explicit EOP (end-of-packet) indication must therefore be inserted into the data stream. This indication must be clearly recognizable so that it cannot be mistaken as another character received from the network.

A break signal (holding the data line in space state for at least a time long enough to send an entire character) leaps to the eye here. A MCU sitting between TP-UART and PC can easily observe the gaps between characters received from the TP-UART. Passing the incoming characters to the PC unaltered, it only needs to insert a break signal whenever it encounters a

gap length that exceeds the threshold. The inter-frame gap on the TP1 medium is long enough to easily accommodate the additional time required for indicating the EOP condition on the host interface, especially if the latter is operated at 19.200 bps.

This is an elegant solution, since the break signal is an out-of-band signal with respect to the TP-UART host protocol. It is transparent to a host application that ignores break conditions. The protocol is not touched in any way; still, no additional signal line is needed.

Unfortunately, not every (software) serial port implementation is able to handle break states. In these cases another method to indicate an EOP condition is required. By using an escape sequence, the indication can be inserted into the character stream itself, achieving maximum compatibility. In our implementation (which also offers signaling by way of break states), the MCU sends the sequence ESC NULL (0x1B 0x00) to the PC to indicate an EOP condition. Any ESC (0x1B) character incoming from the TP-UART is converted into the sequence ESC ESC. If the host interface is operated at 19.200 bps, even incoming frames consisting entirely of ESC characters can still be transmitted with full wire speed.

In [1], an EOP gap duration of “2 to 2.5 ms” is given without further comment. Earlier data sheet versions offered “2 to 2.5 bittimes” as well; this has however been corrected. For this project, more detailed information was required.

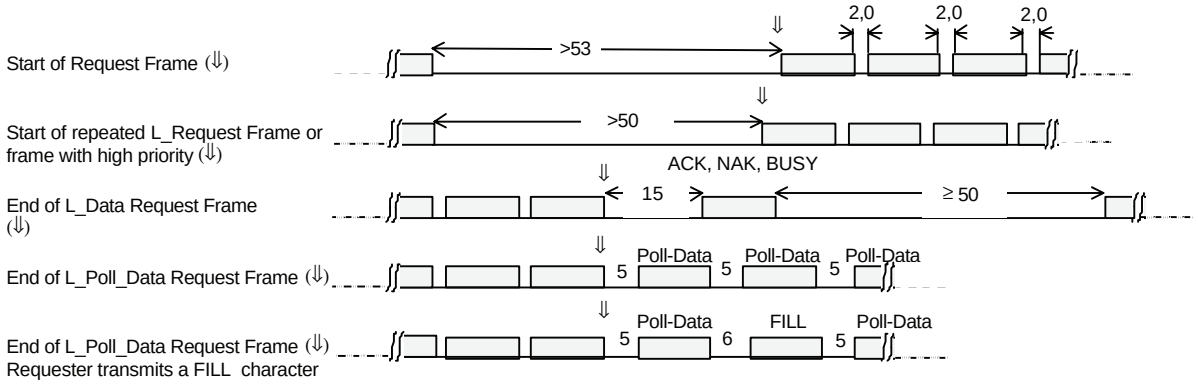


Figure 2: Inter-character timing on TP1 (adapted from [2])

The situation on the TP1 medium is shown in Fig. 2 (gap times are given in medium bit times of 1/9600 s $\approx$ 104 µs). “2 to 2.5 ms” correspond to 19 to 24 bit times, which would mean that only the gaps before request frames are considered EOP conditions (with a considerable safety margin). In this case, especially the upper bound could even be relaxed further. If, however, the gaps before any request frame and the gap before an ACK/NACK/BUSY frame are to be considered EOP conditions, but the gaps before Poll-Data or FILL characters are not, an EOP condition can be defined as a gap of seven bit times or longer. This means that an appropriate indication has to be generated whenever there has been silence on the medium for 7×104 µs = 729 µs immediately following the transmission of a character.

However, the MCU cannot observe the situation on the TP1 medium directly; it depends on the information it receives from the TP-UART. Therefore, it is important to know how the timing of the characters appearing at the TP-UART host interface reflects the timing of the characters on the medium.

While its data sheet does not provide information in this regard, measurements show that the TP-UART appears to propagate characters from the TP1 medium to its host interface with

a constant delay of 1.145 ms (from the first edge of start bit on the medium to the first edge of start bit on the host interface).[1] The MCU can therefore treat periods of silence between characters (more precisely, the relative time between either start bits or stop bits) received from the TP-UART as a one-to-one representation of the situation on the medium.

However, not all characters the MCU receives from the TP-UART have their origin in network traffic. A status character (TP-UART-State.indication/response) can be transmitted spontaneously in case of certain error conditions or in response to a U_State.request by the host controller. In the latter case, the response is usually returned immediately according to our measurements (transmission starts 36 µs after the request has been fully received).

Such a request can also be sent while (or immediately before) a frame is received. The TP-UART data sheet does not provide information when (or if) the response will be returned in this case. However, measurements show that it is held back until the frame has been completely transmitted to the host.

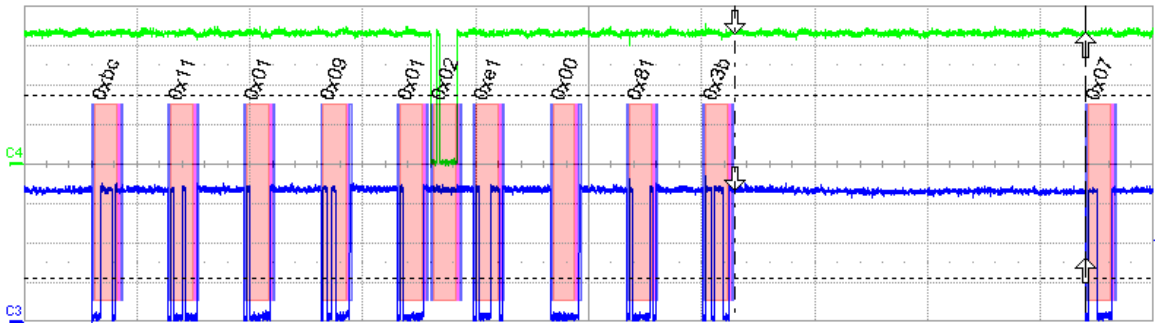


Figure 3: U_State.request during frame reception (2.00 ms/div, 6.22 ms between cursors)

A U_State.request was sent at various time offsets into an ongoing frame reception. One result is shown in Fig. 3. The bottom trace (C3) represents the TP-UART TxD output, the top trace (C4) its RxD input. In all such measurements, the delay measured from the end of the stop bit of the U_State.request (0x02) to the beginning of the start bit of the TP-UART-State.response (0x07) was 6.22 ms. U_AckInformation-Services also sent by the host did not have an influence. Since this delay is far longer than the EOP gap of 729 µs determined above, it is ensured that the TP-UART-State.response cannot be mistakenly interpreted as incoming frame data.

3.2 U_AckInformation timeout condition and handling

According to the TP-UART data sheet, the "U_AckInformation-Service is to indicate if the device is addressed. This service must be send latest 1,7 ms (9600 Baud) after receiving the address type octet of an addressed frame" ([1], p. 12).

Since no further information is provided, this raises the question why the address type octet is referred to, since the TP-UART does not deal with address interpretation at all otherwise. Also, one would expect a timing constraint that is related to the generation of an acknowledgment frame to refer to the end of the corresponding data frame rather than (an offset from) its

[1] All measurements were made using a TP-UART-IC marked as revision "d", with its host interface configured to operate at 19.200 bps.

beginning. In addition, the purpose of specifying a baud rate together with the time period is not obvious.

Therefore, measurements were performed to determine the latest possible point in time the TP-UART will accept a U_AckInformation-Service. A stock KNX device was used to generate a standard data frame on the TP1 medium. An MCU was programmed to transmit a U_AckInformation-Service to the TP-UART at a definable time offset after receiving the L_DATA.ind. This offset was successively increased.

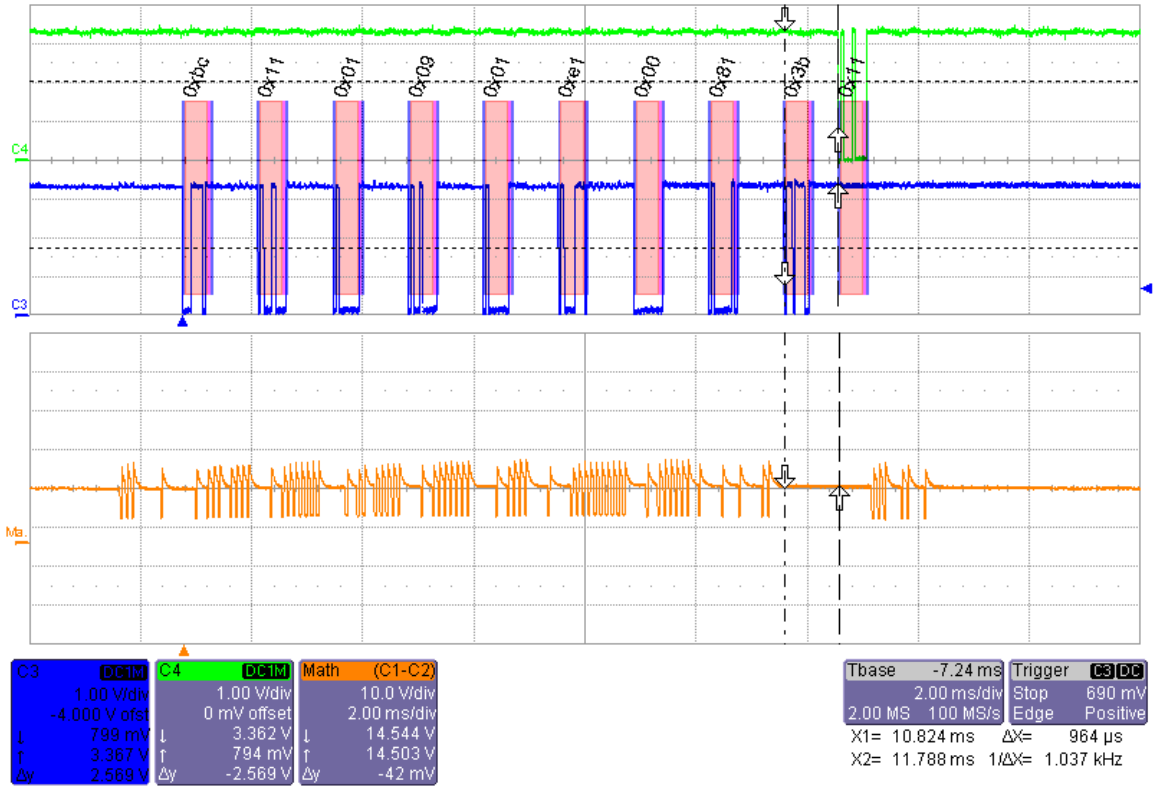


Figure 4: U_AckInformation transmission completed in time

As shown in Fig. 4, completing the transmission of the U_AckInformation-Service 1.54 ms after the TP-UART has started transmitting the check octet is still fine. The middle trace (C3) represents the TP-UART TxD output, the top trace (C4) its RxD input. Note that the TP-UART still puts an acknowledgment frame onto the bus (bottom trace, difference signal between the two TP1 conductors). Actually, the U_AckInformation-Service could not arrive any later, since the acknowledgment frame has to appear on the TP1 medium exactly 15 bit times after the data frame (cf. Fig. 2).

Indeed, if the U_AckInformation-Service is sent 64 µs later, the TP-UART no longer generates an acknowledgment frame (Fig. 5). The sender repeats its transmission three times (not shown in Fig. 5), which means that the U_AckInformation-Service was ignored rather than stored. The TP-UART continues ignoring received U_AckInformation-Services until it has transmitted the next L_DATA.ind. Fig. 6 summarizes the measurements as a sequence diagram, indicating the time slot where the U_AckInformation-Service has to be sent to the TP-UART.

The frame used in these experiments is the shortest possible standard frame containing User-Data (in this case, one Bit). However, a transport layer PDU (e.g., for connection man-

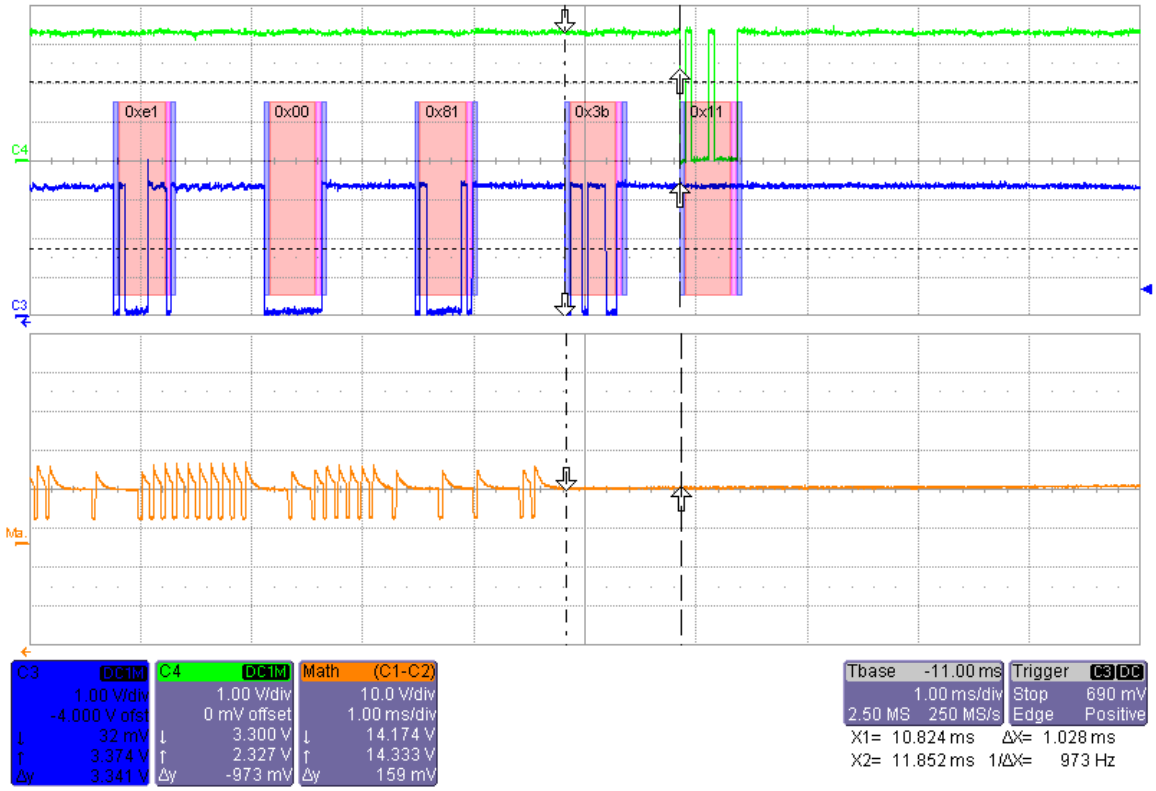


Figure 5: U_AckInformation transmission completed too late

agement purposes) may be one octet shorter still. If a sequence similar to Fig. 6 is considered for such an 8-octet frame and a host interface data rate of 9.600 bps, it turns out that the latest possible moment to start sending the U_AckInformation-Service so that it still reaches the TP-UART in time is quite precisely 1.7 ms after the destination address type/hop count/length octet has been completely received. Very probably, the sentence in the TP-UART data sheet quoted at the beginning of this section refers to this worst-case situation.

It can be assumed that the findings can be also be applied to longer frames than the one tested. Strictly speaking, however, the relaxed response time requirement is an undocumented feature, and no claim can be made about the TP-UART in general. Before relying on it, the time slot allowed by the actual TP-UART chip to be used should be checked. A frame of the maximum expected length must be created on the bus (e.g., using ETS), the MCU programmed to generate the U_AckInformation-Service at the latest point in time as specified above, and the network monitored if an acknowledgment frame is sent by the TP-UART.

Also, it must be considered that when a MCU is placed between the TP-UART and the PC to monitor this time slot, the constraints described apply to its TP-UART side. The maximum allowable response time for the PC is shorter due to the transmission delays between MCU and PC and processing delay in the MCU.

With the precise timing requirements for sending acknowledgment information defined, the question remains how to react when the PC does not meet them. The MCU could of course easily generate a U_AckInformation-Service itself. However, if the interface is to be as transparent as possible (and, thus, stateless), the MCU has no way of knowing if the received frame

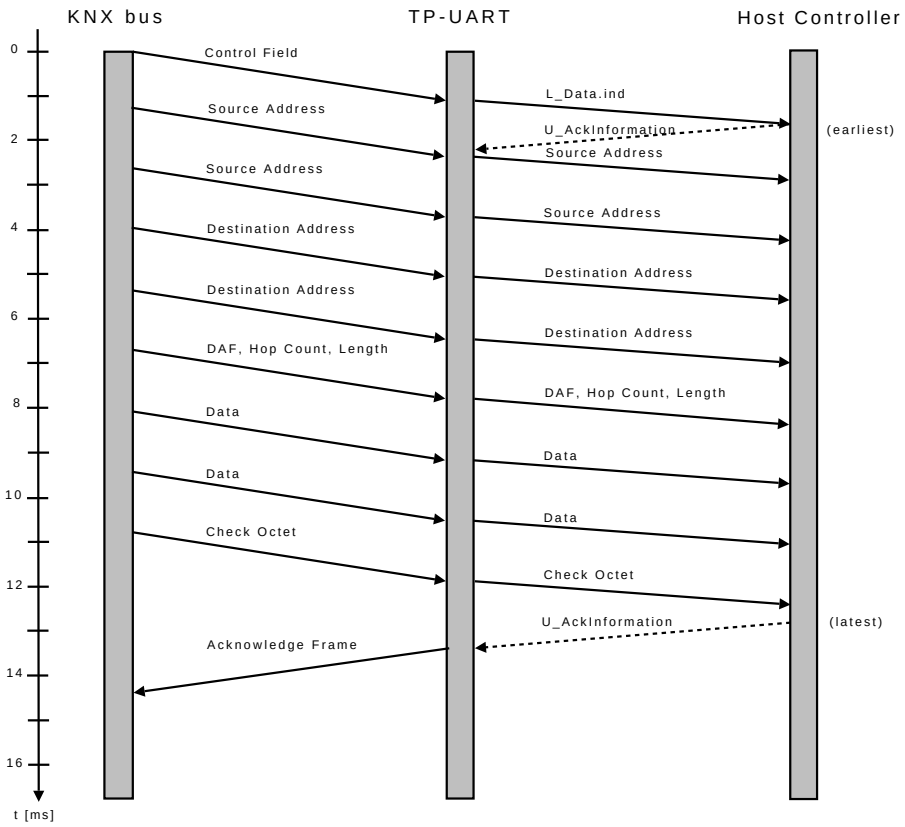


Figure 6: Time frame for U_AckInformation

contained a destination address handled by the PC. Therefore, none of the possible acknowledgment frames (ACK, NACK, BUSY) is appropriate from a semantic point of view.

Not sending an acknowledgment frame, on the other hand, leaves the reason why a node could not find any of its addresses in the received frame open: It may have been because the frame had another destination, but it may also have been because the destination information was destroyed by a transmission error. In the case of our interface, it may also have been because access to the node addresses could not be provided in time. Thus, the current MCU firmware does not send U_AckInformation-Services by itself. This can easily be changed if a different behaviour appears more appropriate.

In particular, one may choose to focus on the effect caused by the different types of acknowledgment frames (or absence thereof): The receiver can send BUSY/NACK to force the sender to retransmit unconditionally (up to the maximum number of repetitions); send ACK to suppress repetitions unless explicitly requested by another node (of the same group) via BUSY/NACK; or remain silent, causing a retransmission unless another node sent ACK. Since the PC application will usually have a large enough buffer to hold – and eventually process – all incoming frames, causing a frame to be sent again probably does more harm than good. Thus, having the MCU cause the transmission of an ACK frame by default may be an alternative worth considering. Actually, couplers do a similar thing when they acknowledge an incoming frame although they cannot know for certain if they will be able to re-send it via the other interface.

Unless an ACK frame is sent by default, a frame retransmission is to be expected when the PC fails to meet the U_AckInformation deadline. The MCU could store the late U_AckInformation

and apply it to the repeated transmission. However, it is hard to see a possible benefit of such a strategy: If the PC is usually fast enough, it should be able to create the U_AckInformation for the repeated L_DATA.ind itself; if it often fails to meet the deadline, but its message buffer is large enough and its average processing capacity sufficient to prevent the buffer from overflowing, sending ACK frames by default seems to be a better solution since it avoids the retransmissions on the network; finally, if its message buffer size and average processing capacity are insufficient, it is very questionable if the time bought by additional retransmissions will have any positive effect at all. On the other hand, implementing this strategy obscures the protocol interface. Also, to be safe, this would not be a matter of only checking the repeat flag, but entire frames would have to be stored and compared. Therefore, the current implementation (like the TP-UART) suppresses late U_AckInformation-Services until it has transmitted the first character of an incoming frame (L_DATA.ind).

In any case, such a timeout situation is best avoided on the PC side. It is therefore essential to inform the user when a timeout has occurred. In the current implementation, this is done by displaying the timeout count via the LEDs on the interface (freezing at the highest number that can be represented in case of overflow).

4 The TP-UART host protocol: Further insights

While the motivation for this project certainly was to find a way of working around the timing issues during frame reception, the necessity of dealing with the TP-UART host protocol in detail also led to some insights on protocol aspects not immediately related to this problem.

4.1 Reaction to corrupted service requests

The TP-UART-State.Indication-Service (as described in the TP-UART data sheet) contains “protocol error” and “receive error” flags. However, the conditions that would provoke such error indications are not revealed.

In order to shed some light on this, selectively corrupted service requests were sent to the TP-UART. All of them were based on the same short standard data frame (in the following, all data are shown in hexadecimal):

```
BC 11 01 09 01 E1 00 81 3B
```

If this frame is to be transmitted via the TP-UART, the required host protocol services have to be added. The inserted characters are ‘80’ (U_L_DataStart), ‘81’ ... ‘87’ (U_L_DataContinue) and ‘48’ (U_L_DataEnd).

```
80 BC 81 11 82 01 83 09 84 01 85 E1 86 00 87 81 48 3B
```

Since this is a correct service request, the TP-UART transmits the frame on the TP1 network and receives it the same time, propagating it back to the PC:

```
BC 11 01 09 01 E1 00 81 3B 8B
```

Its output is identical with the original KNX frame. An L_Data.confirm Service with the “successful” flag set (8B) is appended, indicating a successful transmission.

A corrupted check octet is answered with a TP-UART-State.Indication-Service with the “receive error” flag set (in the following, characters that are different from the initial example are underscored):

To TP-UART: 80 BC 81 11 82 01 83 09 84 01 85 E1 86 00 87 81 48 3C

To HOST: 47

The TP-UART also indicates a receive error when its RxD input remains silent for about 3 ms after a U_L_DataStart service (80).

If the request is sent with its control byte being corrupted, the TP-UART returns two State.Indication-Services, with the protocol error flag set in the first (17) and the receive error flag set in the second one (47).

To TP-UART: 80 7C 81 11 82 01 83 09 84 01 85 E1 86 00 87 81 48 3B

To HOST: 17 47

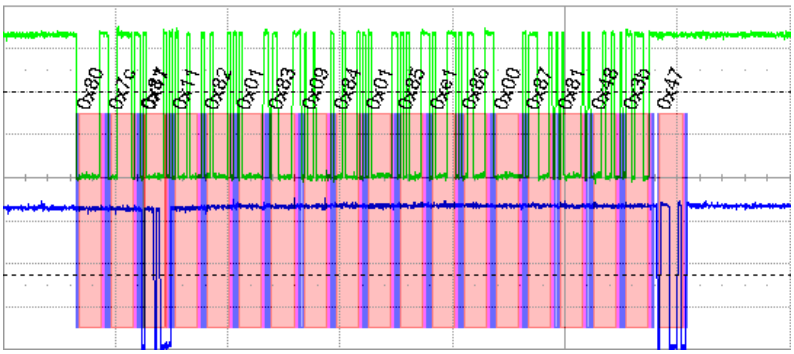


Figure 7: Corrupted control byte producing error indications

Fig. 7 shows the corresponding waveforms (top: TP-UART RxD, bottom: TP-UART TxD). An interesting detail to see is that the protocol error is generated immediately after the receipt of the corrupted control byte. The receive error is generated after receiving the check octet (possibly due to the mismatch caused by not adapting the check octet to match the altered frame).

The count field of the U_L_DataContinue services separating the KNX frame bytes must be successively incremented by one. If this condition is violated, no error indication is given. Rather, the content of the KNX frame is altered:

To TP-UART: 80 BC 81 11 82 01 BE 09 81 01 81 E1 81 00 81 81 48 3B

To HOST: BC 81 01 09 01 E1 00 81 AB 8B

Although the host can detect the error condition by comparing the intended frame data with those echoed back by the TP-UART, the changed frame has already been transmitted on the network. This is acknowledged by an L_Data.confirm service (8B). Surprisingly, the wrong check octet does not cause an error but is modified as well.

The penultimate byte sent to the TP-UART is the U_L_DataEnd service. In case this one is corrupted, the TP-UART adapts the frame length according its new value. In this example, its count field is increased by one. In the resulting KNX frame one byte is inserted and the checksum is adapted by the TP-UART. A positive L_Data.confirm service is returned.

To TP-UART: 80 BC 81 11 82 01 83 09 84 01 85 E1 86 00 87 81 49 3B

To HOST: BC 11 01 09 01 E1 00 81 22 19 8B

Likewise, if the count field is increased by seven, seven bytes are inserted. Again, successful transmission is indicated by a L_Data.confirm service.

To TP-UART: 80 BC 81 11 82 01 83 09 84 01 85 E1 86 00 87 81 4F 3B

To Host: BC 11 01 09 01 E1 00 81 22 FF A8 F7 59 FF D0 CF 8B

However, the count field of U_L_DataEnd cannot be increased without limits. Its maximum value is 7F, which corresponds to the 64 octets telegram send buffer of the TP-UART. If this value is exceeded (in the following example, it is set to 80), the TP-UART indicates a protocol error.

To TP-UART: 80 BC 81 11 82 01 83 09 84 01 85 E1 86 00 87 81 80 3B

To Host: 17

It would also be interesting to know how the TP-UART deals with faulty frames received from the network (e.g., containing wrong check octets, timing violations, or parity errors). Although suitable hardware (TP-UART in analog mode) is available, this was not yet possible within the scope of this project for organizational reasons.

4.2 Reset behaviour

After power-up and following a “warm start” in response to a U_Reset.request (0x01), the TP-UART transmits a TP-UART-Reset.indication (0x03). However, 0x3F or even nothing at all is often received instead. This can be explained by examining closely what happens on the TP-UART TxD output.

In normal mode, TxD is High when idle. On power-down, TxD falls Low. On power-up, TxD and RESn are kept Low until Vcc has stabilized. TxD goes back to idle level approximately 5 ms after RESn has been released (Fig. 8).

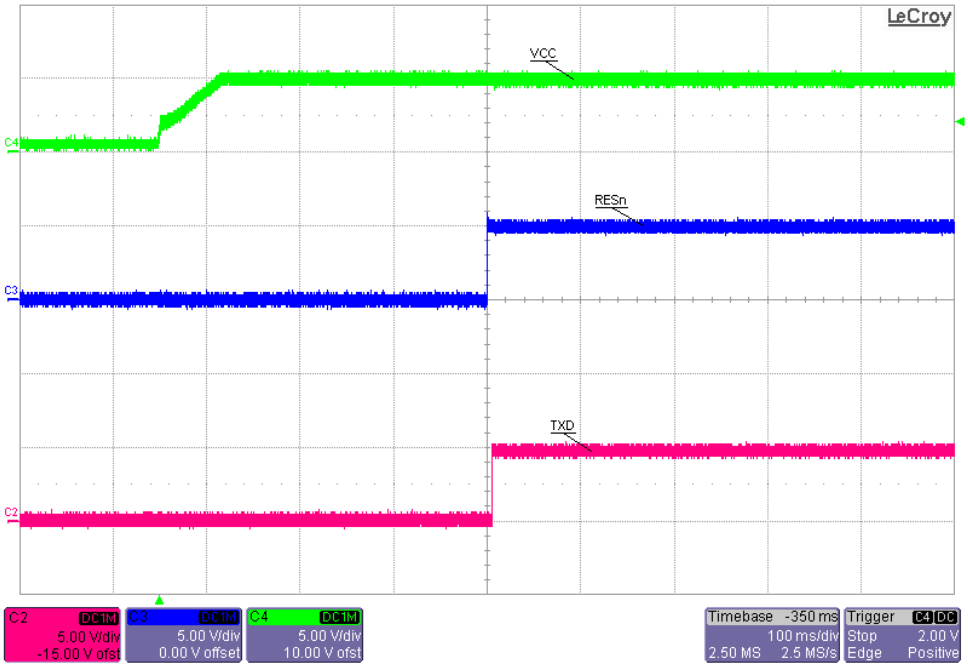


Figure 8: TP-UART power up

Upon reception of a U_Reset.request service, TxD immediately falls Low. This state persists for 5.4 ms. Fig. 9 shows the corresponding waveforms (top: TP-UART RxD, bottom: TP-UART TxD).

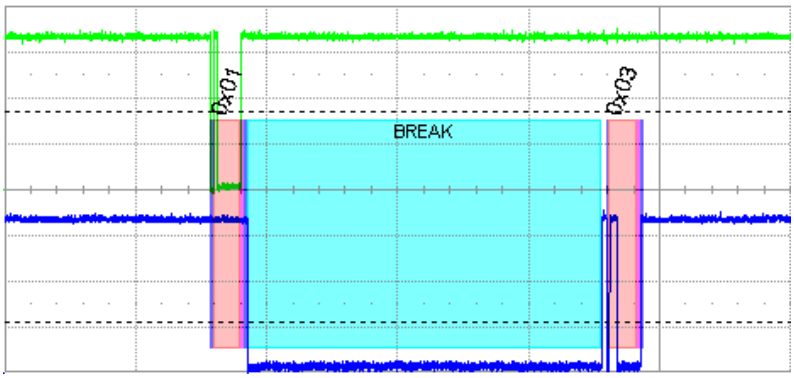


Figure 9: TP-UART software reset (2.00 ms/div)

In both cases, TxD remains Low for an extended period of time. This corresponds to a break condition. Depending on the PC configuration, this is often propagated to the application as 0x00.

The TP-UART-Reset.indication is transmitted immediately following this long break state, with TxD rising to idle state only for a single bit time (Fig. 10). If the serial port handling code on the PC “wakes up” too late, this character can be lost or distorted. For example, if the penultimate data bit is interpreted as a start bit and parity checking is deactivated, the application may see 0x3F instead.

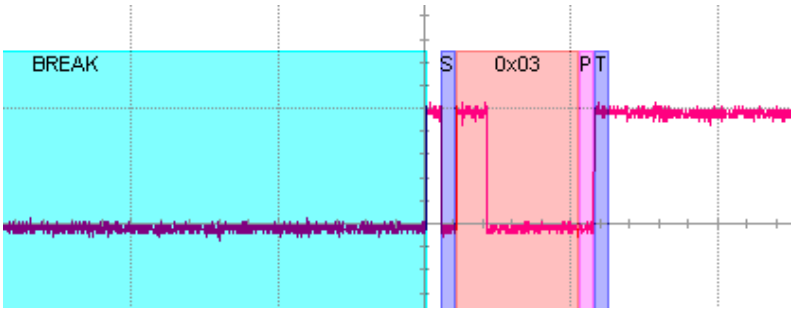


Figure 10: TP-UART-Reset.indication (500 µs/div)

5 Summary and outlook

The TP-UART host interface protocol requires precise timekeeping to detect an end-of-packet condition and quick reaction if acknowledgment frames are to be sent. Closer examination reveals that U_AckInformation services may in many circumstances be sent significantly later than the TP-UART data sheet specifies. Moreover, protocol properties with regard to error handling were demonstrated that may be considered unexpected.

Strategies for handling end-of-packet indication and U_AckInformation timing constraints to make the TP-UART host protocol suitable for PCs have been discussed. They rely on a microcontroller sitting between TP-UART and PC. This microcontroller, however, only minimally interferes with the original protocol. The strategies were implemented and tested using a small

TI MSP430 family microcontroller. Existing PC applications can be adapted with minimum effort. Future versions of eibd [12] will also provide support for the enhanced protocol.

A KNX PC interface based on this microcontroller and the TP-UART was presented. Its versatile design allows it to be used for tasks far beyond the scope of this project. It has already been tested. Its design will be made openly available.

The PCB was designed using the EAGLE Layout Editor. Since the PCB dimensions do not exceed 80 $\times$ 100 mm, changes can be made using EAGLE Light Edition, which is distributed free of charge for non-commercial and evaluation purposes. The toolchain required to program the used microcontroller is available without cost as well.

A test and demonstration program for the PC side was created as well. It uses termios to access the serial port and can be compiled using GCC without changes on both Linux and Windows/Cygwin. It allows to send and decode TP-UART host protocol services and can – of course – deal with the end-of-packet indication methods described previously. It also provides access to the RS-232 DTR, RTS and CTS signals. Like the microcontroller firmware, it will be made available as open source.

The interface hardware allows many further ideas to be implemented. For example, escape sequences could be used to carry basically any information to the PC, including the SAVE signal. The MCU may be used to improve error handling or could delay the TP-UART-Reset.indication for slow PC serial port implementations.

A USB version could easily be created by replacing the level conversion/parasitic power supply circuitry with an integrated USB to UART chip. Until then, stock USB to RS-232 converters can be used.

References

[1] Siemens EIB-TP-UART-IC Technical Data,
http://www.automation.siemens.com/et/gamma/download/tpuart.pdf

[2] KNX Handbook, Vol. 3, Part 2, Ch. 2, System Specifications: Communication Media: Twisted Pair 1, v1.0 AS

[3] Schematic of the Siemens-TPUART-interface for the serial port,
http://os-projects.fh-deggendorf.de/images/tpuart-interface.png
(retrieved in November 2001, no longer available online)

[4] TP-UART interface by C. Troger,
https://www.auto.tuwien.ac.at/downloads/eib4linux/tp-uart-interface.zip

[5] Disch Systems TP-UART Interface,
http://disch-systems.de/download/tpuart_interface_en-datasheet.pdf

[6] Siemens Bus Transceiver Module PCB Technical Data,
http://www.opternus.com/uploads/media/BTM_PCB_datasheet_V2.0.pdf

[7] F. Praus, W. Kastner and G. Neugschwandtner, “A versatile networked embedded platform for KNX/EIB”, KNX Scientific Conference 2006, Vienna.

[8] R. Stocker and A. Grzemba, “Linux Device Driver for the TP-UART interface”, EIB Scientific Conference, October 2001, Munich.

[9] W. Kastner and C. Troger, “Interfacing with the EIB/KNX: A RTLinux device driver for the TPUART”, Proc. 5th IFAC Intl. Conference on Fieldbus Systems and their Applications (FeT ’03), pp. 29–36, 2003.

[10] TP-UART Linux kernel driver updates by R. Buchinger and M. Kögler, https://www.auto.tuwien.ac.at/a-lab/eib4linux.html

[11] Disch Systems EIB Driver for Linux, http://disch-systems.de/download/edrv_en-datasheet.pdf

[12] `eibd` daemon homepage, http://www.auto.tuwien.ac.at/~mkoegler/index.php/eibd

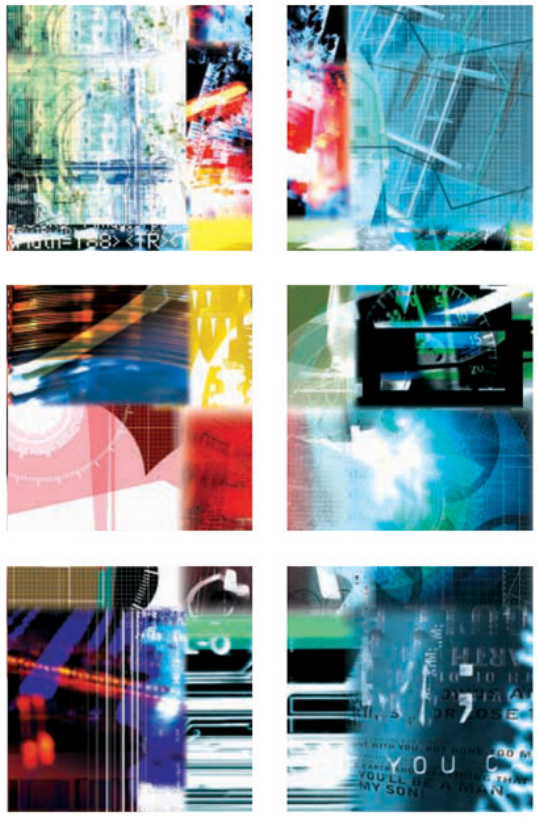
[13] United States Patent 7,291,938, “Power supply apparatus and method based on parasitic power extraction”, 2007.

[14] TI MSP430x12x Mixed Signal Microcontroller (Rev. C) Datasheet, http://www.ti.com/lit/gpn/msp430f123

[15] TI MSP430F11x2, MSP430F12x2 Mixed Signal Microcontroller (Rev. D) Datasheet, http://www.ti.com/lit/gpn/msp430f1232

[16] TI MSP430x1xx Family User’s Guide (Rev. F), http://www.ti.com/litv/pdf/slau049f

All online references last accessed October 2008, unless otherwise noted.



www.knx.org

User applications development using embedded Java

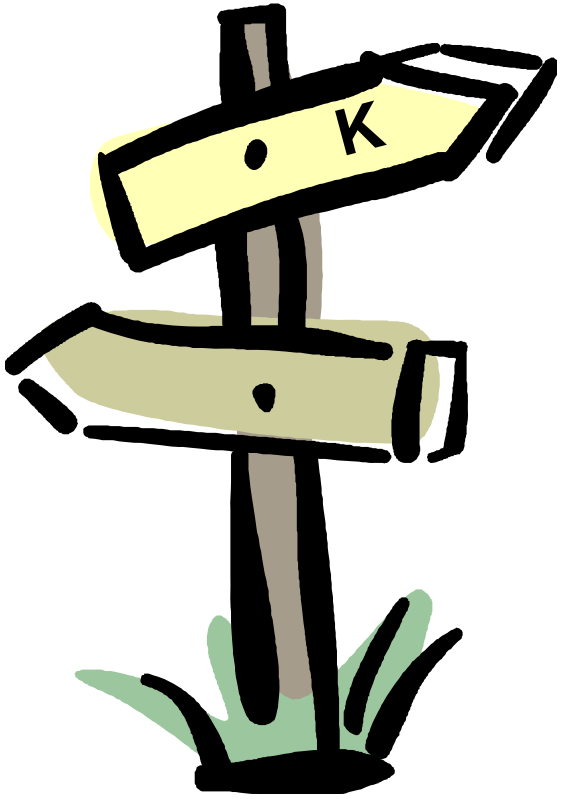
AUTOMATION SYSTEMS GROUP

Vienna University of Technology

F. Praus, W. Kastner

Overview

- **User application (UA) development**
  - State of the art
- **Concept**
- **Approach**
  - System software
  - Sandbox
  - Management tool
- **Implementation**
  - Hardware
  - Software
- **Evaluation**



User application development

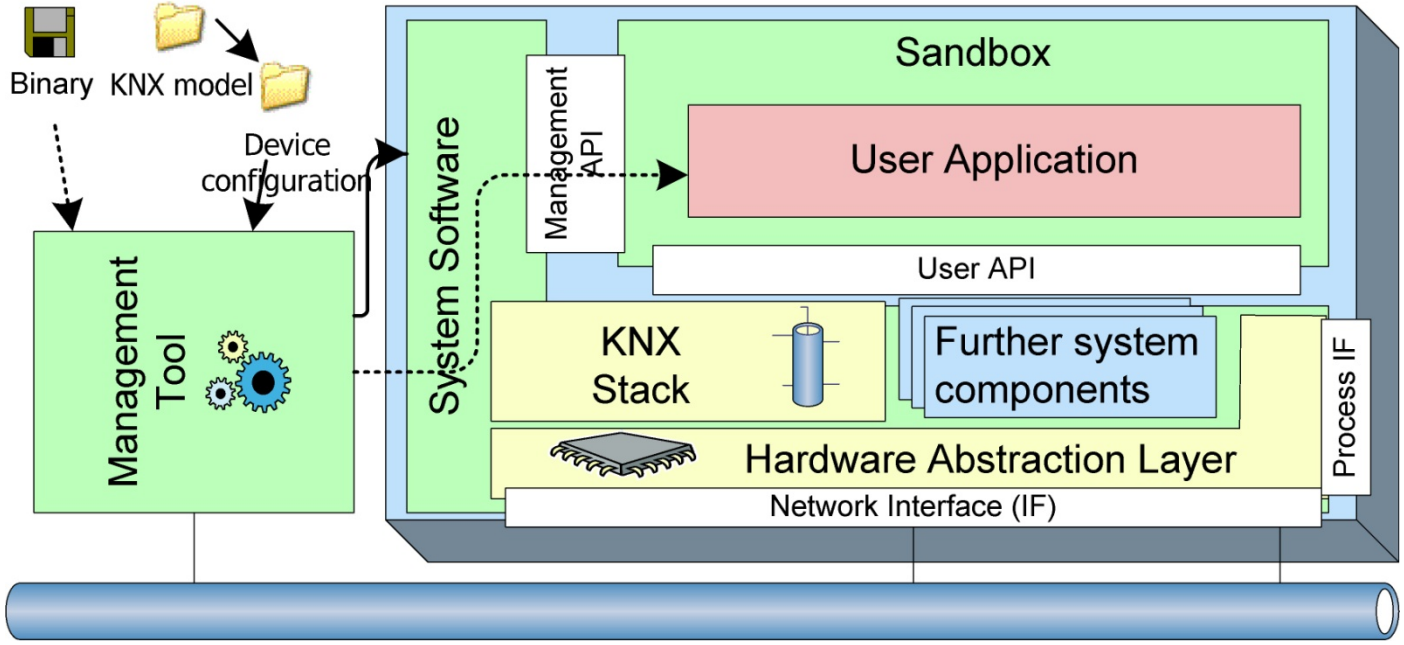
- **BCU1, BCU2**
  - Limited memory, no high level programming language, hardware specific issues
- **NEC based BIMs**
  - C programming language, debugging
- **TP-UART**
  - Dedicated communication stacks
- **Windows**
  - Falcon
- **Linux**
  - Drivers, Calimero

Concept

- **Open source framework supporting**
  - Rapid innovation and implementation
  - Configuration
  - Deployment and execution
- **Low end, low cost, low power KNX devices**
  - <100KBytes memory, <25MHz CPU speed
- **Separate UA from system software to allow**
  - High level programming language (Java)
  - Hide KNX protocol details
  - Portability

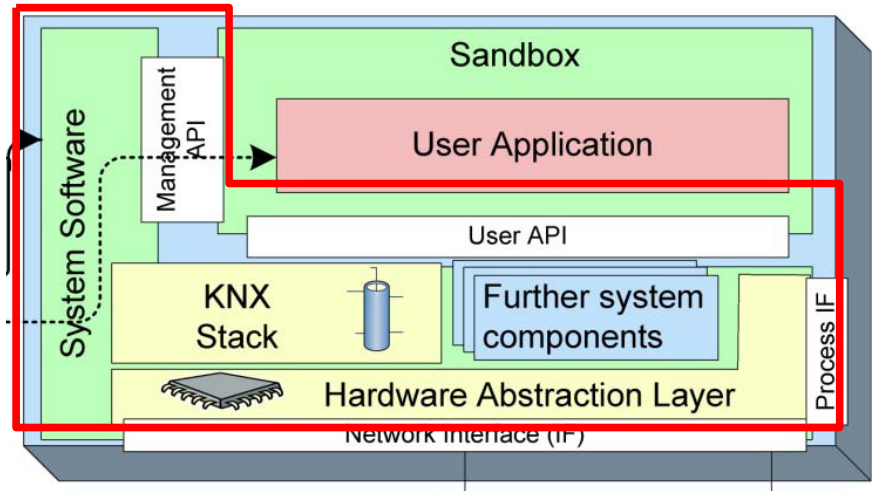
Approach

- System Software
- Sandbox
- Management Tool



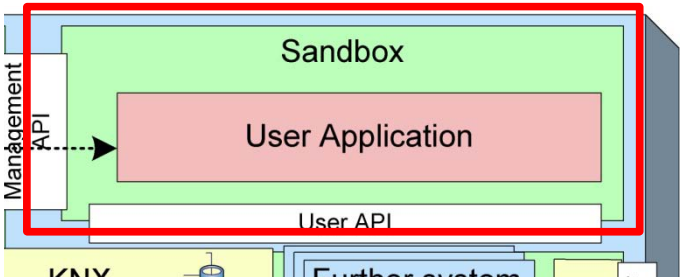
Approach – System Software

- **Hardware abstraction layer**
  - Hide hardware specifics
- **Network protocols**
  - e.g., KNX stack
- **User API**
  - Interface between system software and UA
- **Management API**
  - Interface to management tool



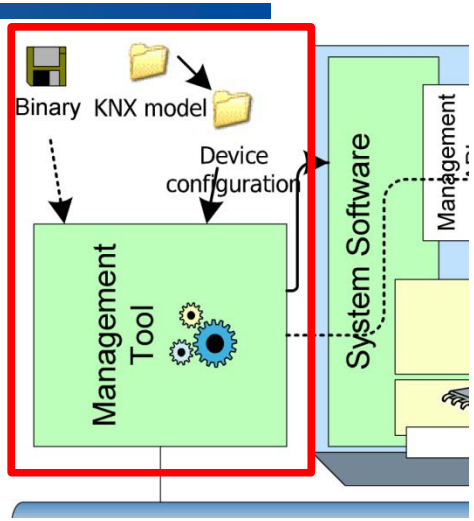
Approach – Sandbox

- **Downloaded programs executed in controlled way**
  - e.g., Java Micro Edition, TinyVM, NanoVM
- **Support rapid development**
- **Designed for little memory and low overhead**
  - Some limitations



Approach – Management tool

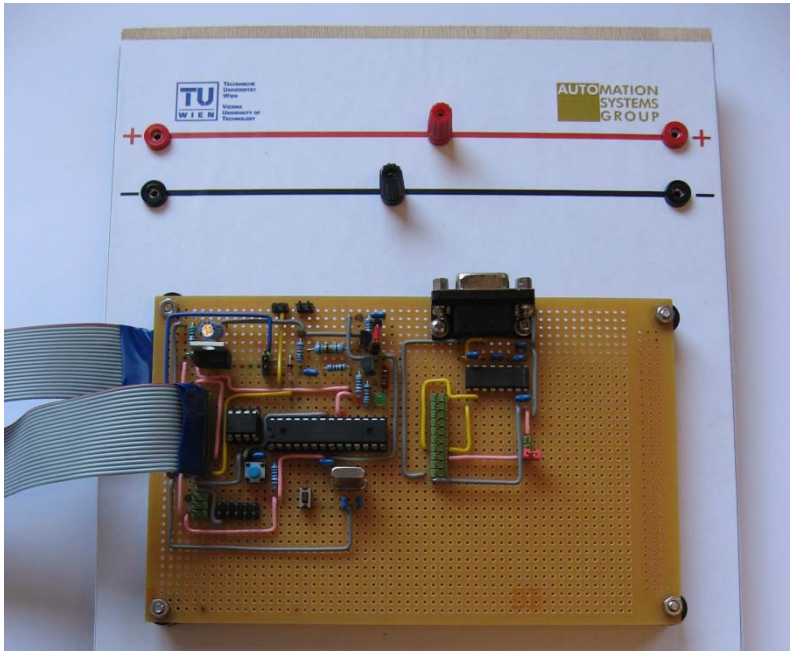
- **Generic KNX model**
  - Communication endpoints
    - KNX group objects
  - Common encoding
    - KNX datapoint types
  - Functionality
    - KNX functional blocks
- **Device configuration**
  - References KNX model
- **Management tool**
  - Process configuration data
  - Deploy UA




```
<DatapointTypes>
        <DPT id="1.001" Name="DPT_Switch" DataTyperef="1">
                <Encoding valueref="0" description="Off"/>
                <Encoding valueref="1" description="On"/>
        </DPT>
        [...]
</DatapointTypes>
[...]
<FunctionalBlock ObjectType="417" Name="FB_Light_Actuator"
                Title="Light Actuator">
        <Functional_specification>Switch the light according to the
            received value on the OnOff datapoint [...]
        </Functional_specification>
        <DataPoints>
                <Output>
                        <DP id="1" Name="InfoOnOff" Abbr="IOO"
                                Mandatory="true"
                                Description="To send the actual
                                    state of the light"
                                DPTref="1.001"/>
                </Output>
                <Input>
                        <DP id="2" Name="OnOff" Abbr="OO"
```

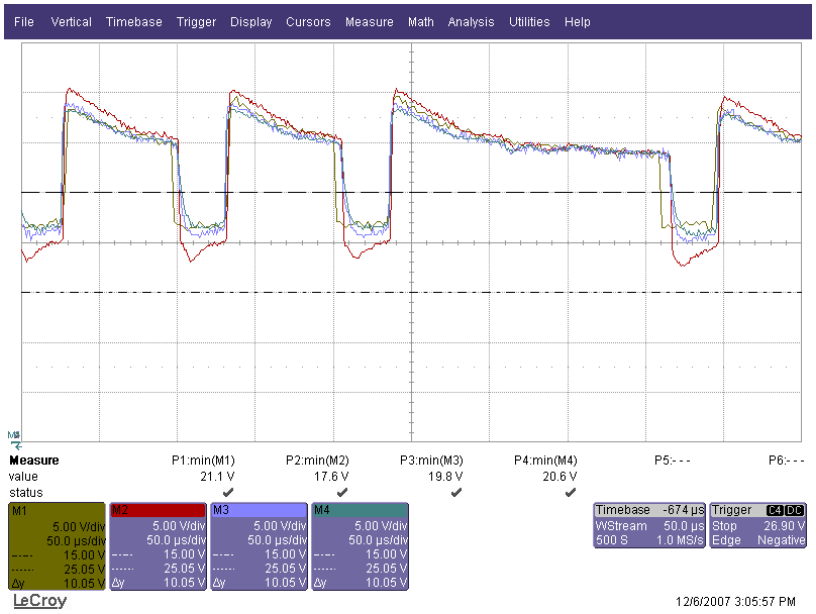
Implementation - Hardware

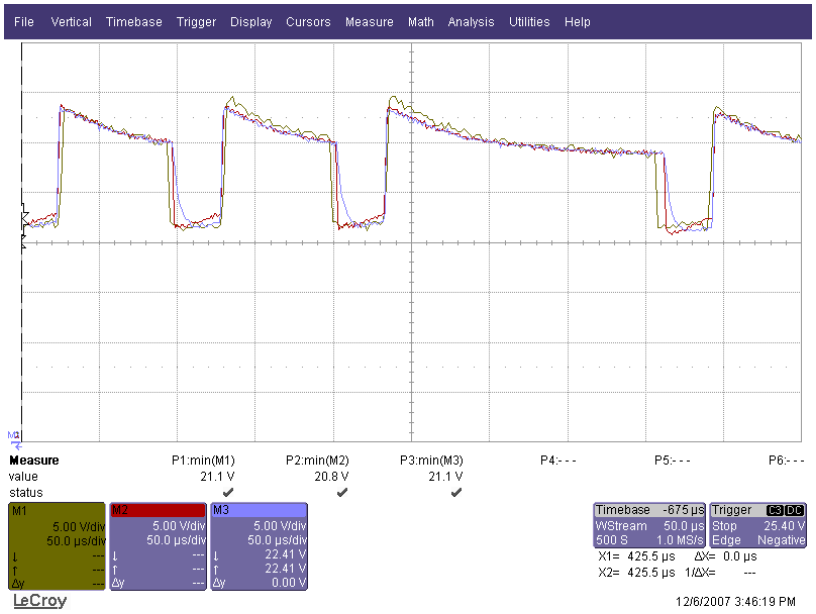
- **Implementation on two hardware architectures**
  - Control KNX compliant devices
- **SAC stripboard**
  - Atmel ATMega168 microcontroller @ 8MHz, 3.3V
    - 16KB flash, 512B EEPROM
    - attached 2KB external EEPROM for program storage



Implementation - Hardware

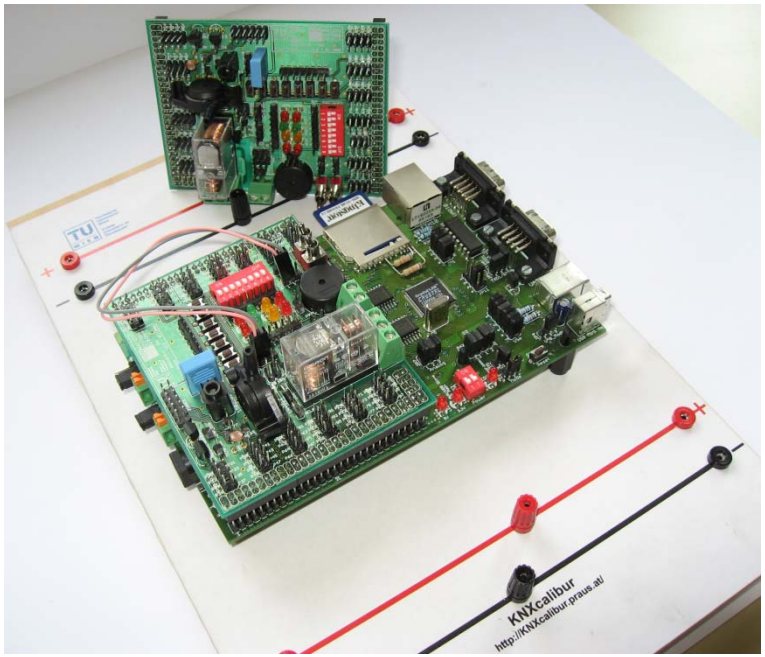
- **SAC stripboard using Freebus basic circuit to access KNX bus**
  - Free and cheap home automation system
  - Compatible to KNX systems ?





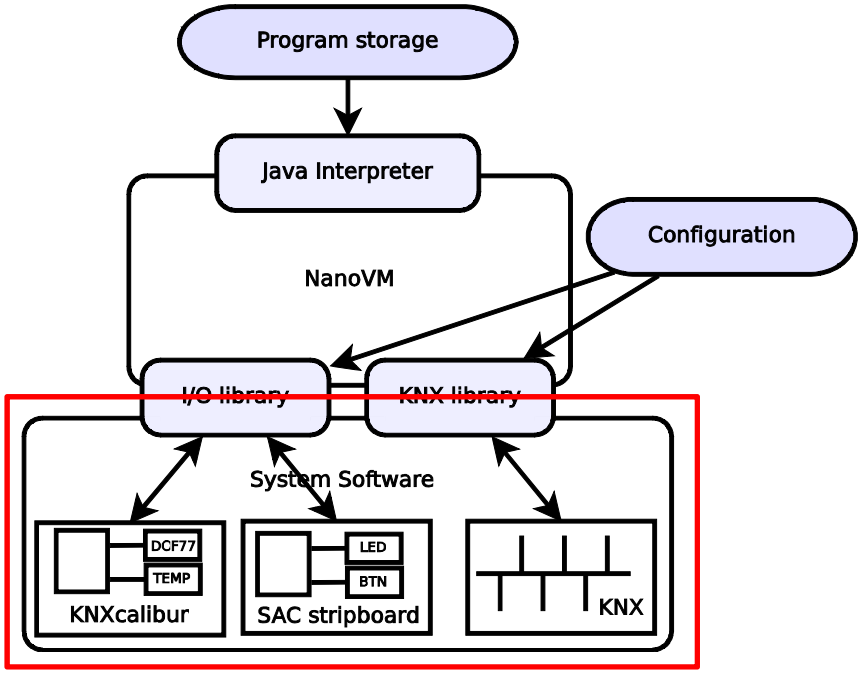
Implementation - Hardware

- **KNXcalibur**
  - As presented on KNX Sci06
  - Daughterboard extension
    - LEDs, buttons, switches
    - Temperature sensor
    - Humidity sensor
    - Pressure sensor
    - Light dependent resistor
    - DCF77



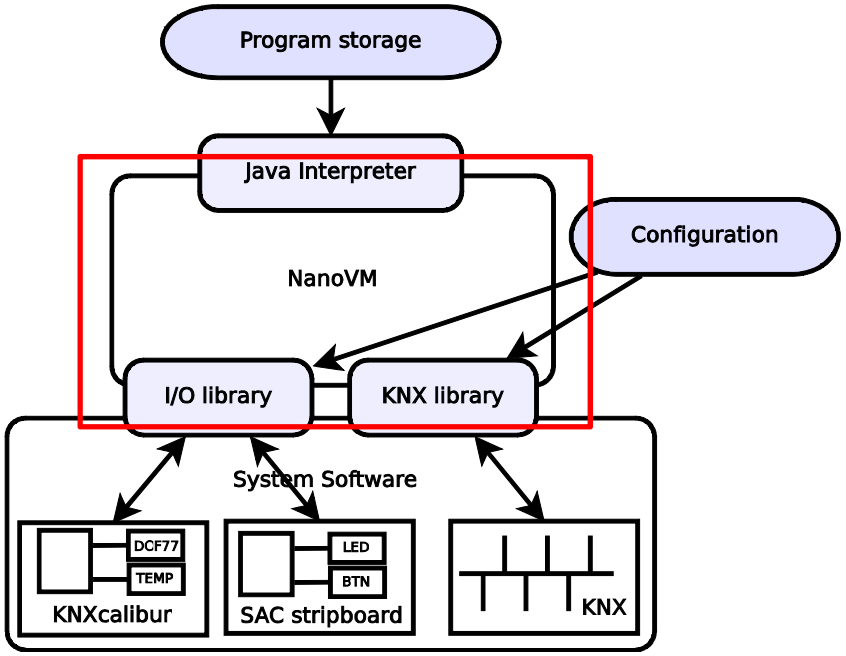
Implementation - Software

- **SAC stripboard and KNXcalibur**
  - Modular, using HAL
- **System Software**
  - KNX stack
    - Bit stuffing and TP-UART driver
    - Group communication
    - Management communication
  - Peripheral drivers



Implementation - Software

- **NanoVM for sandboxing**
  - VM providing subset of Java class library
    - Integer & Float arithmetic, inheritance, garbage collection, …
  - Fetches UA instructions from external storage
  - Executes UA



Implementation - Software

- **User API**
  - KNX library
  - I/O library

```
private Vector<GroupAddress> associatedGroupAddresses;
[..]

public boolean hasChanged(); // Returns the state of the group
   object. Returns true if the group object was changed since the
   last call of goChanged

public DataPointType readValue(); // Reads and returns the current
   value of the group object from the memory. The operation does
   not cause any messages to be sent.

public void WriteValue(DataPointType value); // Writes the value of
    the group object. If group addresses with write access are
   associated with the group object, the value is sent to the bus.

public void RequestUpdate(); // Requests updating the group object.
    Causes sending of a read message for each associated group
   address with write access.
```

```
<DeviceConfig xmlns:devconf="http://[...]">
        <ProgramID>1</ProgramID>
        <KNXLibrary>
                <IndividualAddr>1.1.1</IndividualAddr>
                <GroupAdresses>
                        <GroupAdress id="ga1">0/0/1</GroupAdress>
                        <GroupAdress id="ga2">0/0/2</GroupAdress>
                </GroupAdresses>
                <FunctionBlock id="417">
                        <GroupObjects>
                                <GroupObject id="go1">
                                        <DataPoint dpref="1" garef="ga1"/>
                                        <DataPoint dpref="2" garef="ga1"/>
                                </GroupObject>
                        </GroupObjects>
                        <Parameters>
```

- **Configuration**
  - XML based
  - References to generic model
  - Conversion tool

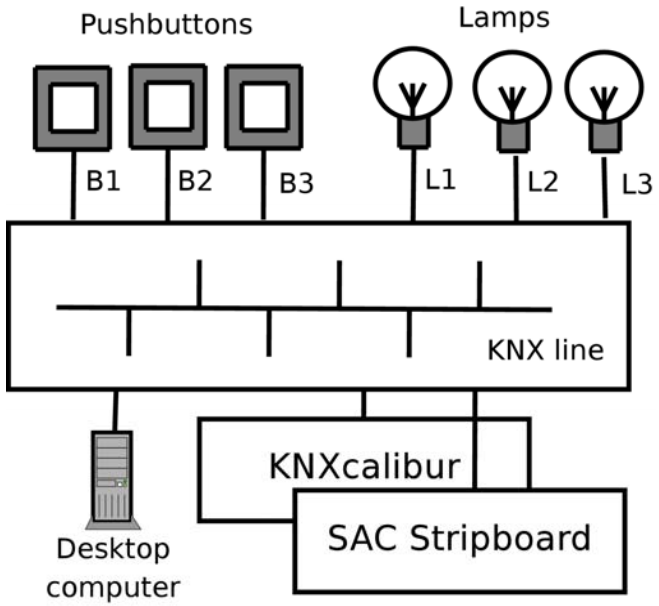
Development workflow

- **System software**
  - Compilation using MCU specific toolchain
- **UA**
  - Development using standard Java distributions
  - Definition of XML based configuration
  - NanoVMTool
    - Stripping class files
    - Configuration
    - Upload
- **Example program**
  - Reacts to changes of group object 1

```
package examples.sebas;

import nanovm.IOLib.io;
import nanovm.KNXLib.*;

class GroupObjects {
  KNXGroupObject go1=new KNXGroupObject("go1"); // configuration is
      automatically read using the id "go1"

  public static void main(String[] args) {
    System.out.println("Group objects example");

    while(true) {
      if (go1.hasChanged()) {
        DataPointType dp = go1.readValue();
        io.led(1,dp.intValue());
      }
    }
  }
}
```

Evaluation

- **Test environment**
  - Standard KNX components
  - SAC stripboard
  - KNXcalibur
- **Memory usage on SAC stripboard**
  - ~8/16 Kbytes flash
  - 512 Bytes internal and 2 Kbytes external free
  - ~400/1000 Bytes SRAM



Evaluation

- **Acceptable performance for simple building automation tasks**
  - Light control
  - PID controller
- **Matrix multiplication**

Matrix dimension	Native C	Java (NanoVM) int. EEPROM	Java (NanoVM) ext. EEPROM
2	31 μ s	13.2 ms	216 ms
4	440 μ s	20ms	1.34 s
8	3.92 ms	595 ms	9.60 s
16	32.80 ms	4.64 s	-

Conclusion

- **Promising architecture**
- **Demonstrated feasibility and usefulness**
- **Familiarization with KNX eased**

- **Outlook**
  - Ease configuration
    - Tool
    - ETS export
  - Extend VM
    - Exceptions
    - Threads

User applications development using embedded Java

Fritz Praus Wolfgang Kastner *

Automation Systems Group
Institute of Automation
Vienna University of Technology

Treitlstraße 1-3, A-1040 Vienna, Austria
{fpraus,k} @ auto.tuwien.ac.at

Building Automation Systems (BAS) aim at improving control and management of mechanical and electrical systems in buildings. It is the task of Sensors, Actuators and Controllers (SACs) to interact with the physical environment and perform measurement and control tasks. In building automation networks they, typically, consist of a network interface and a microcontroller (MCU) where an application specific hardware is attached. In a similar way, the software may be split into a system software part providing basic functionality and a (customizable) user application (UA) dealing with the attached hardware. Within KNX this concept is well known, as customized application modules are connected via the physical external interface to standardized bus attachment units. However, KNX UA development traditionally required profound hardware knowledge.

This paper presents an approach to leverage user application development. With limited hardware resources of SACs in mind, an embedded virtual machine provides a lean environment for Java based user applications. Interfacing to KNX and the specific hardware is done by a well-defined Java user application programming interface. In addition, a management interface allows to download and configure UAs inside the virtual machine. The programming concept, configuration and management issues as well as the workflow are described in detail. As proof of concept for SACs supporting our approach two hardware platforms have been chosen: a stripboard equipped with an ATMega168 MCU and KNX connection via the Freebus project, and KNXcalibur, a Fujitsu MCU based board connected to KNX via TP-UART. The contribution is rounded up by a first performance analysis.

*This work was funded by FWF (Österreichischer Fonds zur Förderung der Wissenschaftlichen Forschung; Austrian Science Foundation) under the project P19673.

1 User application development – State of the art

To develop KNX TP devices different possibilities exist, which for example differ in flexibility, certification effort or costs per device:

BCU 1, BIM M111 and BIM M115 based devices feature a MC68HC05B6 or compatible type CPU running at 2 MHz with 176 Bytes RAM, 256 Bytes EEPROM and 5936 Bytes ROM. BCU 2 and BIM M113 use a MC68HC05BE12 CPU running at 2.4576 MHz. They have 384 Bytes RAM, 991 Bytes EEPROM and 11904 Bytes ROM ([1] Part 9/4). Although the tool software ETS allows customization of applications developed for these bus attachment units (BAUs), their first time creation remains quite complex due to the following reasons:

- The hardware does not provide enough memory for advanced user applications (UA). Often memory optimizations have to be performed by hand.
- No toolchain is available, that offers high level programming languages (e.g., C, C++). Therefore UAs have to be developed relying on assembler code[1].
- The UA developer has to consider MCU specific issues since a comprehensive hardware abstraction layer is missing (e.g., RAM flags).

To overcome the limitations of rather out-dated BAUs and ease development, a new generation of BIMs has been developed. They are based on the NEC 78K0/Kx2 MCU and offer from 8 KBytes up to 48 KBytes flash memory. Additionally, UAs can be developed in the C programming language and debugged using the IAR Embedded Workbench. Moreover, a dedicated evaluation board exists [3].

For development of UAs on a separate MCU, the BIMs and BCUs offer a high-level access to the network stack via the PEI, using it as an (asynchronous) serial interface. Nevertheless, in such cases interfacing the KNX with a TP-UART IC is a more attractive alternative [4]. It handles the physical and most of the data link layer and can be connected to various MCUs for further processing of KNX frames on dedicated communication stacks.

To access the network stack of devices by various interfaces from Microsoft Windows based systems, a common high-level application programming interface (API) ("Falcon") is commercially available. For the Linux operating system open source kernel drivers for accessing BCUs and TP-UART based serial interfaces exist [5]. KNX access via the Java programming language is possible using Calimero [6].

2 Concept

The traditional development of UAs is not a very straightforward task and requires profound knowledge about KNX. To remedy this situation, the goal of the project is to create an open-source framework for developing UAs for low end ($<$100 KBytes memory, $<$25 MHz CPU speed) KNX devices. Application designers shall be supported to allow rapid innovation and implementation, configuration, deployment and execution of arbitrary (uncertified) software. The solution shall be low cost and not require any special hardware modification, thus allowing easy and

[1] For the sake of completeness it has to be mentioned, that possibilities exist to ease the user application development for BCUs [2].

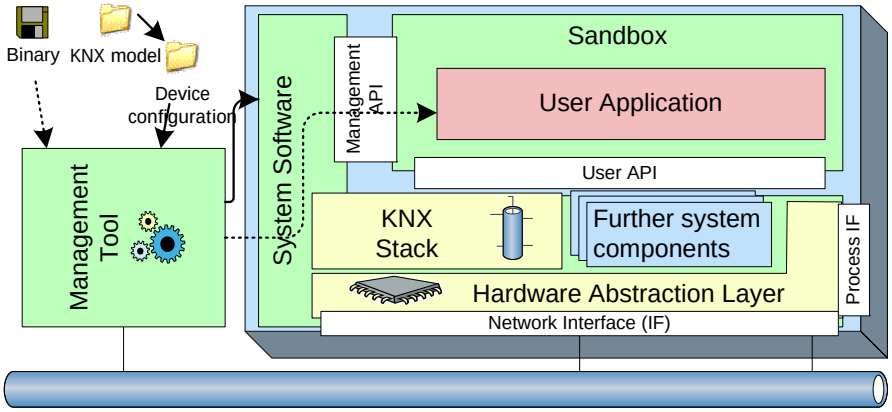


Figure 1: Architecture

compatible integration into KNX systems. The idea is to separate the system software running on the device from the application programs as well as the node configuration. This model encapsulates the system software entities in a way which is inspired by the object-oriented paradigm. It also makes use of KNX functional blocks to describe the application behavior. The following list describes the intended advantages the proposed framework provides:

- The framework supports the use of high-level languages, in particular Java. Thus the desired behavior can be obtained more easily.
- UA development is simplified, since the application programmer does not have to cope with details concerning the KNX protocol or the system software.
- It can be ensured that all (bus-)communication is standard compliant. Since the UA is separated from the system software, the latter has to be certified only once.
- Portability of UAs can be achieved due to their separation from the system software.
- UAs can only issue a defined set of operations and may not interfere with the system software, which eases a possible certification process.
- Limitations to operations a UA issues can be enforced by the system software.
- Home and building automation (HBA) control tasks can be carried out very flexible and with an adaptable configuration. Information invisible and unaccessible to the UA can be stored and used on the system (e.g., configuration parameters).
- Applications which are easier to understand are also less error prone.

3 Approach

The proposed framework consists of three main parts (cf. Figure 1):

- A system software provides controlled access to system resources. It encapsulates hardware specific details, the network protocol stack, the process interface as well as any further system components and offers clean interfaces for the UA (Section 3.1).

- A sandbox executes the UA in a controlled way and is also designed to support its rapid development. It interfaces the system software and provides a clear abstraction of the underlying hard- and software by providing an object-oriented access using the Java programming language. The application designer can thus focus on the application development. This also allows portability of applications between devices offering the same sandbox (Section 3.2).
- A management tool is used to configure the parameters of the system software (e.g., network address) and deploy the UA into the sandbox. Configuration is based on the standardized functional blocks (e.g., light actuator defined in [1] Part 7/20), which are stored in a generic way. A device specific configuration file references this knowledge base and provides the necessary additional information (Section 3.3).

3.1 System software

The system software consists of various layers and intends to provide building blocks which can be mixed and matched to support different hardware configurations. It is designed to maximize code reuse. A change in the combination of the software modules or a change in the hardware design should only require a minimum of modification to the software. Besides taking care of initialization tasks and managing the available resources of the system, the system software consists of the following parts:

- To access the hardware in an independent and modular way at the lowest level of the system software, a Hardware Abstraction Layer (HAL) hides the peculiarities of basic I/O handling and on-chip peripherals. It allows to easily deploy the developed software to other microcontroller architectures allowing flexibility in design and to fulfill the differing resource requirements.
- Network protocol stacks such as IP (and related protocols) and KNX can be integrated according to the requirements of a particular application and are along with further system components (e.g., for controlling peripherals) located on top of the HAL. Interaction with these components and the system software is exclusively possible via two well-defined application programmer interfaces (APIs).
- The user API serves as an interface between the system software and the UA and provides various services (e.g., network access, access to on-chip peripherals such as timers, process interaction). Besides allowing to control the possibilities of a UA such a generic API also increases portability and compatibility of applications on different platforms. The design of the user API will vary for different application fields. Nevertheless, it should be kept as general as possible and in the ideal case, the user API should operate at a very high abstraction level. For example, in the case of networking, only valid incoming messages shall be reported to the UA, while erroneous receptions will automatically be discarded and negatively acknowledged. This way, the UAs could be kept simple and at the same time can concentrate to perform intended actions, only.
- The management API interfaces with a management tool, which pre-processes the binary of a UA as well as its corresponding configuration and allows access to the system software to support the total replacement and download of UAs. Moreover, this tool allows

customization of an application by adapting runtime parameters for the sandbox defined in a configuration file.

- Besides, the system software runs the sandbox and manages its required memory.

3.2 Sandbox

According to [7], "a sandbox is a technique by which a downloaded program is executed in such a way that each of its instructions can be fully controlled". It is often used to execute untrusted programs or untested code with the essential benefit that the system outside the sandbox is protected from (malicious) actions by the UA. Additionally, the behavior of the program in the sandbox can be monitored and controlled (e.g., via a watchdog). As an extension to sandboxing, the concept of a virtual machine (VM) is used to execute so called applets. Applets are self-contained programs which run in the context of other programs (e.g., web browsers).

There are also a number of Java virtual machines available for embedded systems. Sun offers the official Java Micro Edition (Java ME) [8] for embedded devices, targeted at mobile phones, personal digital assistants (PDAs) and similar. Although strictly reduced in size and overhead, it still features powerful programming interfaces, robust security and built-in network protocols. But due to this relatively comprehensive feature set, the minimum system requirements specify the need for 160-512 KBytes of total memory and a 16 Bit processor clocked at 16 MHz or higher. Especially the memory requirements are slightly beyond the limitations of our targeted hardware.

Besides the official Java versions, there is a number of implementations available which are targeted at systems with very low resources. These implementations often come at the price of only offering a subset of the full Java functionality. Only two such solutions shall be presented here: NanoVM [9] and TinyVM [10]. Both virtual machine implementations allow the execution of standard Java bytecode, at least after preparation with an automated tool. They target very low-profile embedded systems. The NanoVM was originally written for the Atmel ATMega8 microcontroller included in the Asuro robot, and has a memory footprint of approximately 7 KBytes. The TinyVM operates on Lego Mindstorms RCX programmable bricks which are equipped with an Hitachi H8 microcontoller. It has memory requirements of around 10 KBytes. Both VMs have some limitations regarding the provided Java language features. For example the NanoVM does not support multithreading. The TinyVM is generally more advanced (of course also due to the more powerful hardware it runs on), and supports multithreading, exceptions and synchronization but for example misses floating point operations. But despite these limitations both VMs have their possible applications, since the omitted functionality is often not required for their intended use.

Within our proposed concept we specify an execution environment for UAs. This sandbox executes the UA in a controlled way and offers services via the user API, the latter requires to fulfill its assigned task. It thus also supports the rapid development of UAs. It is obvious that such a sandbox has to be designed for little memory usage and low overhead. While this approach may at first seem inappropriate for a low-end embedded system due to the relatively high resource requirements of such techniques, it offers outstanding possibilities. Besides, the resource requirements can be lowered to a significant extent with the acceptance of certain limitations. The sandbox does not need to support fully fledged programming models, since the desired operations, especially in control tasks, are often quite simple. For such purposes,

UAs more or less consisting of a sequence of simple operations may be sufficient which can be supported by a resource-saving sandboxing implementation.

3.3 Management Tool

Within KNX the communication endpoints relevant for process data exchange are referred to as group objects, which can be regarded as application related variables exposing particular aspects of the functionality of a KNX node. Group objects can be data sources, which provide information to other devices, or data sinks, which carry out certain actions according to the information received. By assigning group objects of multiple nodes to a common logical group a network-wide shared variable can be formed, which is being held consistently by the KNX network stack. The common encoding used to ensure that the shared value will be interpreted in a consistent way is referred to as data point types (DPT) (e.g., DPT 1.001 stands for a Boolean value with the state labels On and Off). Yet, the expressiveness of DPTs with regard to describing processing rules (i.e., how the change of one data point affects others) is limited. For this reason, the functionality of the application is described by declaring one or more functional blocks (e.g., light actuator).

To ease managing and configuring of devices, we defined KNX relevant configuration data (e.g., DataPointTypes, KNX functional blocks) in a machine readable form (cf. Listing 1), which forms the basis for configuration within our proposed concept. A device specific configuration file references the defined functional profiles and provides the necessary additional information (e.g., mapping of group addresses). The management tool is used to parse and combine the relevant data and configures the system software running on the KNX device. Furthermore, it is used to preprocess the binary of the UA and to download it into the sandbox.

Listing 1: Part of model defining functional profiles

```
<?xml version="1.0" encoding="UTF-8"?>
<KNX xmlns:xsi="http://[...]">
        <DataTypes>
                <DataType id="1" name="Boolean" format="1" unit="-">
                        <Range>
                                <Value id="0">0</Value>
                                <Value id="1">1</Value>
                        </Range>
                </DataType>
                [...]
        </DataTypes>
        <DatapointTypes>
                <DPT id="1.001" Name="DPT_Switch" DataTyperef="1">
                        <Encoding valueref="0" description="Off"/>
                        <Encoding valueref="1" description="On"/>
                </DPT>
                [...]
        </DatapointTypes>
        [...]
        <FunctionalBlock ObjectType="417" Name="FB_Light_Actuator"
                        Title="Light Actuator">
                <Functional_specification>Switch the light according to the
                    received value on the OnOff datapoint [...]
```

```
                </Functional_specification>
                <DataPoints>
                        <Output>
                                <DP id="1" Name="InfoOnOff" Abbr="IOO"
                                        Mandatory="true"
                                        Description="To send the actual
                                           state of the light"
                                        DPTref="1.001"/>
                        </Output>
                        <Input>
                                <DP id="2" Name="OnOff" Abbr="OO"
                                        Mandatory="true"
                                        Description="To switch the light On
                                             or Off"
                                            DPTref="1.001" />
                                [...]
                        </Input>
                        <Parameter>
                                <DP id="3" Name="Timed Duration" Abbr="P1"
                                        Mandatory="false" Description="
                                           Duration to switch..."
                                        DPTref="7.005"
                                        Default="0" />
                                [...]
                        </Parameter>
                </DataPoints>
        </FunctionalBlock>
        [...]
</KNX>
```

4 Implementation

4.1 Hardware

In the effort to develop a comprehensive development environment for BAS, we evaluated our concept with focus on its applicability to low-end BAS nodes (i.e., SACs). The presented software approach has been deployed on two different hardware architectures and the proof-of-concept implementation is able to control a KNX compliant light (on, off).

4.1.1 SAC stripboard

To gain first experimental results and evaluate the imposed overhead and performance, a low cost hardware platform based on a stripboard has been assembled (cf. Figure 2, [11]). It is on the one hand intended to be powerful enough to handle the presented sandbox approach but on the other hand to stay very small scale and represent the lowest end SAC devices. While it appeared reliable during development and testing of the software running on it, it is not intended for real-life use.

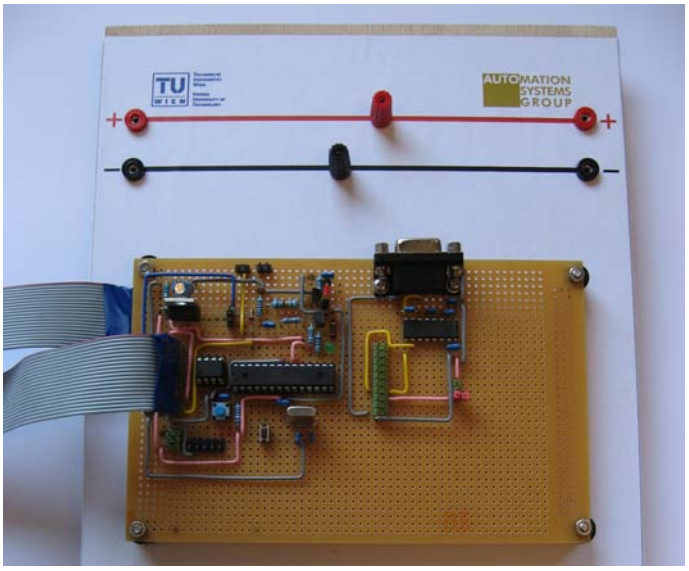


Figure 2: SAC stripboard

We chose an Atmel ATMega168 microcontroller clocked at 8 MHz, featuring 16 KBytes flash, 1 KByte SRAM, 512 Bytes EEPROM. Among others, the controller provides 23 programmable I/O lines, two 8-bit Timer/Counters and one 16-bit Timer/Counter with various compare modes, a programmable serial USART, a byte-oriented 2-wire serial interface and two external interrupts. Some external components have been equipped to provide additional functionality: For debugging, testing and basic interfacing a LED and a pushbutton are attached. An additional external EEPROM (MICROCHIP 24LC16B/P) with a memory size of 2048 Bytes is used to provide extra non-volatile memory. It is connected via the 2-wire serial interface supported by the ATMega168, which is actually an I^2C connection (the naming is different due to licensing reasons). The connection is clocked at 400 kHz, which is the fastest possible with the external EEPROM. It still limits the data rate significantly, since there is some overhead contained in the required data exchange protocol. A MAXIM MAX 3232CPE dual channel driver/receiver chip is equipped to convert controller signals to RS-232 levels to enable serial communication, for example with a PC. On the side of the microcontroller, it is directly connected to the USART pins of the ATMega168.

To access the KNX bus, a slightly adapted version of the basic circuit of the Freebus project [12] has been considered to convert the MCU signals to KNX signals and to power the device using the bus line. Freebus is an open source project maintained by hobbyists which basically aims at providing a free and affordable system for home automation, which is compatible to KNX bus systems. The developers try to build devices with similar functionality as it is provided by retail manufacturers while using only a minimum number of parts to remain as cheap as possible. Within our test environment we evaluated the interoperability of the hardware with standard KNX devices regarding the signal levels. Although the basic circuit generally works as promised, early tests showed a slight problem. A KNX bus line holds a voltage of almost 30 V when idle. This state corresponds to a logical one. To signal a logical zero, a device may pull down the bus voltage to a minimum of 19 V. The problem observed in the tests occurs when the basic circuit is used to signal a zero. In this case the voltage is pulled down to 16 V which is too low according to the KNX specification.

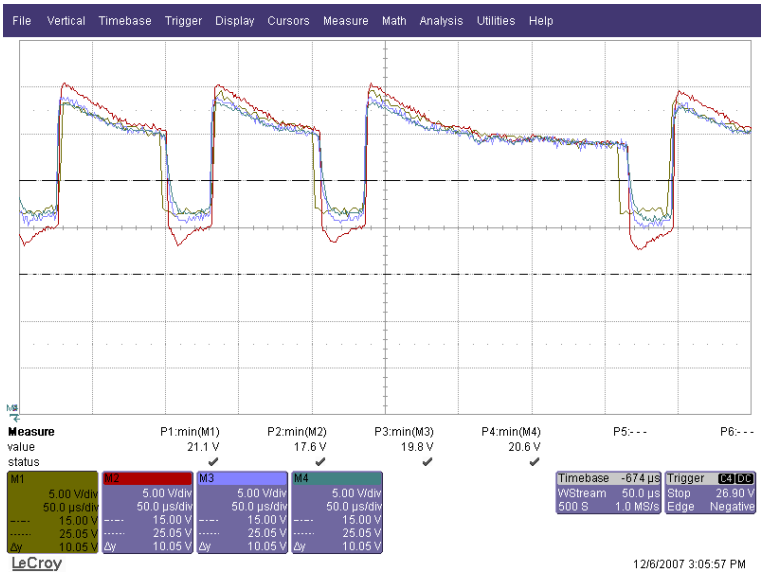


Figure 3: Voltage characteristics comparison

Figure 3 shows some measurements taken with the original basic circuit and with some modifications. A voltage characteristic as produced by a retail device (in this case a pushbutton by Siemens) is shown in yellow. The red characteristic is produced by the SAC stripboard. The difference in the minimum level is obvious (there are also slight timing differences which can be neglected in these measurements). As a first approach to fix this issue, it was tried to limit the voltage drop by using a series resistor at the base of the 2N7000 MOS-FET. The blue characteristic was recorded with a $15k\Omega$ series resistor and green with $20k\Omega$. It can be seen that with the series resistors, the voltage drops only to the desired levels but at cost of the steepness of the edges.

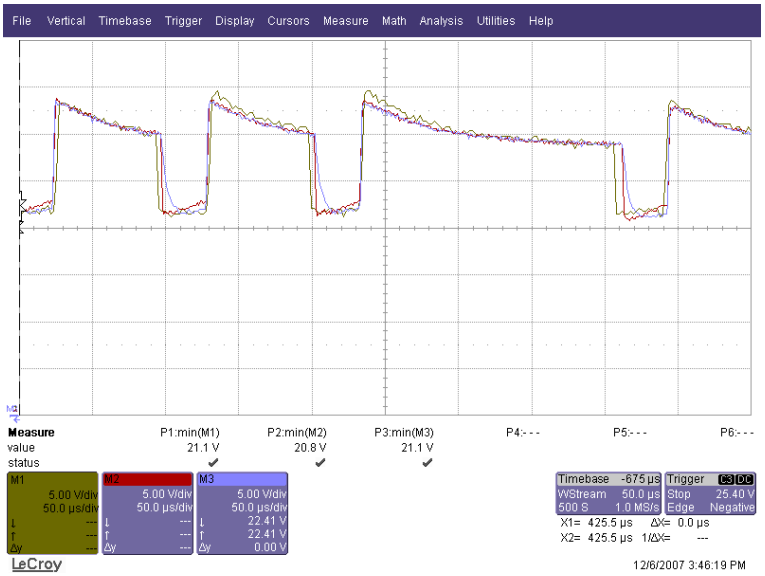


Figure 4: Voltage characteristics comparison

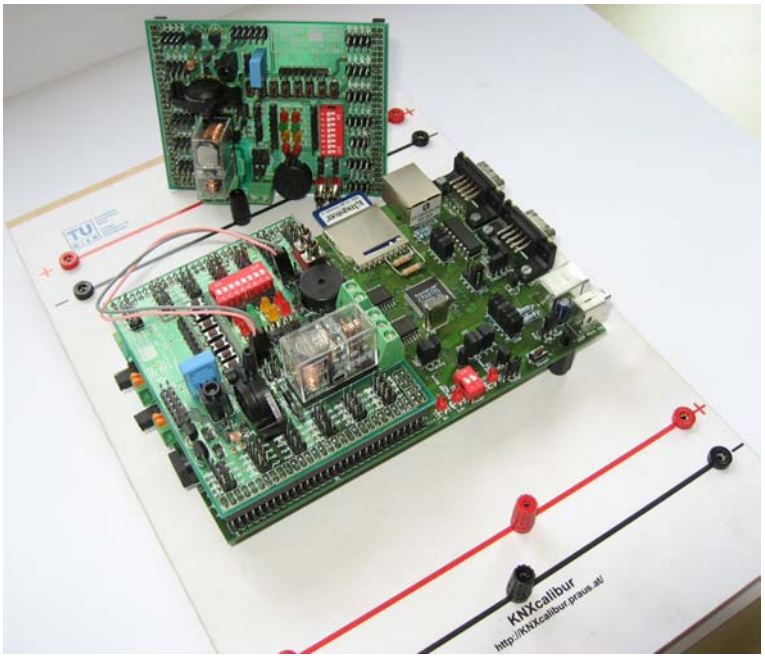


Figure 5: KNXcalibur

Better results were obtained without series resistors by just lowering the operation voltage of the microcontroller. In Figure 4, measurements taken with such a setup are shown. The yellow curve again shows the characteristic of a retail device. Here, the red characteristic is taken with an operation voltage of 3.1 V, the blue one as comparison with a series resistor of $20k\Omega$ and 3.2 V operation voltage. The red characteristic looks very much like an original one, with decent steepness of its edges.

However it has to be noted, that the circuit without any modifications was used for the major time of experimenting with the SAC stripboard. And although the signal characteristics fall slightly out of the specifications, no problems with the transmission of signals could be observed.

4.1.2 KNXcalibur

KNXcalibur [13] is based on a Fujitsu 16 bit MB90330 family MCU. It operates at a maximum frequency of 24 MHz and provides 24 Kilobytes RAM, 384 KB flash, 4 UARTs, a 8/10 Bit A/D converter and SPI (Serial Peripheral Interface) as well as an external bus interface similar to the ISA bus. Moreover, USB functionality and Ethernet connectivity via the Cirrus Logic CS8900A Ethernet LAN controller are realized. To support persistent storage of large amounts of data on KNXcalibur a SD/MMC card connection has been integrated. Connection to KNX/EIB is realized with the Siemens TP-UART IC. Via the available pin headers an extension daughterboard has been connected. It provides 8 push buttons, 8 LEDs, 8 switches, 1 relais, 1 buzzer and a MAX3471 for RS485 connectivity. In addition the following sensors have been integrated: an HSP15 humidity sensor, a MPX4250 pressure sensor, a NSL19M51 light dependend resistor, an LM335Z analog temperatur sensor, a DS1820 digital temperatur sensor and a DCF77 connector for attaching a FSZ01020 antenna.

4.2 Software

The software has been implemented in a very modular way according to the concept presented before in order to allow flexibility and different hardware configurations. The main basis for hardware indepentent software modules forms the HAL, which has been implemented in an event based way. Using the HAL it is now possible to implement generic modules, such as the KNX network stack: It provides bit stuffing support for controlling the freebus hardware as well as a TP-UART driver for more simplified KNX network access. Irrelevant to this underlying OSI layer 1/2 the KNX stack offers KNX group communication as well as group object handling and management communication via the implemented mangement API (e.g., setting network addresses or parameters). Besides, drivers for the sensors mentioned before are contained in the system software within further software modules. The system software also manages configuration and stores configuration parameters in non-volatile memory.

4.2.1 NanoVM

For sandboxing the UA the NanoVM was chosen, since a port to the AVR MCU already existed and only a port to Fujitsu MCUs had to be done. Some of its features are 15/31 bit integer arithmetic, floating point operations, garbage collection, simple application upload, inheritance, a unified stack and heap architecture and about 20k Java opcodes per second on a 8 MHz AVR. Nevertheless considerable modifications have been applied to achieve the targeted level of functionality. First, the UA is not stored in the flash memory of the MCU but on external storage (external EEPROM in case of the stripboard, SD card for KNXcalibur). Thus the NanoVM had to be modified to allow fetching the UA instructions from external program storage. In addition, Java libraries have been written to access the peripherals (via the user API) of the hardware platforms and to enable KNX compatible data exchange.

Figure 6 outlines the flow of information between the software components. UA Java bytecode instructions are fetched from the program storage and are interpreted by the Java interpreter. The instructions may be operations on internal Java variables or calls to library methods. The library methods read possible configuration values, execute the designated functions on the hardware platform or communicate with peripherals.

4.2.2 User API

The user API forms the main component for the UA developer and is the only way to interact with the hardware. Native Java classes have been defined which form the interface for the UA. The NanoVM maps these Java methods and variables to their C implementations. Currently two libraries are implemented:

The KNX library hides KNX specific details and provides a high-level access to group objects. Communication is solely performed by reading and writing these objects (cf. Listing 2). Any messaging related tasks such as frame and checksum generation and checking, handling of acknowledgments and retransmission in case of errors are carried out by the library.

Listing 2: KNX library

```
class KNXGroupObject {

        private Vector<GroupAddress> associatedGroupAddresses;
```

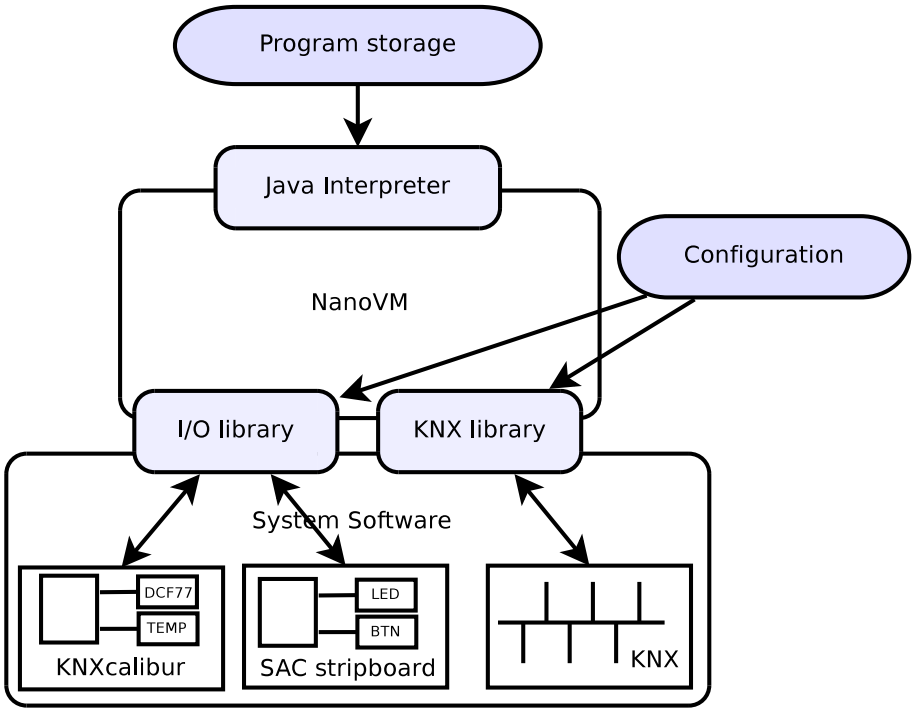


Figure 6: Sequence diagram of the software components

```
        [..]

        public boolean hasChanged(); // Returns the state of the group
            object. Returns true if the group object was changed since the
            last call of goChanged

        public DataPointType readValue(); // Reads and returns the current
            value of the group object from the memory. The operation does
            not cause any messages to be sent.

        public void WriteValue(DataPointType value); // Writes the value of
             the group object. If group addresses with write access are
            associated with the group object, the value is sent to the bus.

        public void RequestUpdate(); // Requests updating the group object.
             Causes sending of a read message for each associated group
            address with write access.
}
```

The I/O library (cf. Listing 3) handles access to peripherals and is, as mentioned, common to every hardware configuration. In case of missing hardware components, the corresponding functions return -2 so that the developer can handle this situation.

Listing 3: I/O library

```
class IOLib {

        // functions marked with (*) return -2 if not available on hardware
```

```
            and -1 in case of any other errors
        public static int led(int led, int value); // 1 turns on LED, 0 off
            , (*)
        public static int btn_is_pressed(int btn); // returns 1 if button
            pressed, 0 if not, (*)
        public static int switch_is_on(int sw); // returns 1 if switch is
            turned on, 0 if not, (*)
        public static int relais(int on_off); // 1 turns relais on, 0 off,
            (*)

        public static int initTempAnalog(void); // initialize analog
            temperature sensor, (*)
        public static Float getTempAnalog(void); // returns temperature
            value

        public static int initTempDigital(void); // initialize digital
            temperature sensor (*)
        public static Float getTempDigital(void); // returns temperature
            value

        public static int initPressure(void); // initialize pressure sensor
             (*)
        public static Float getPressure(void); // returns pressure value

        public static int initLDR(void); // initialize LDR, (*)
        public static Integer getBrightness(); // returns brightness value

        public static int initHumidity(void); // initialize humidity sensor
            , (*)
        public static Integer igetHumidity(); // returns humidity value

        public static int initBuzzer(); // initialize buzzer, (*)
        public static void buzzer(int value); // 1 turns on buzzer, 0 off

        public static int initDCF77(void); // initialize DCF77, (*)
        public static Date getTime(); // returns current date and time

        public static int initRs485(ReceiveListener rhandler); //
            initialize RS485, (*), ReceiveListener gets called in case of
            received frames
        public static String Send2Slave(int slave, int code, String msg);
            // sends "msg" with "code" to "slave" and returns returnmessage
        public static void TellMaster(int code, String data); // prepares "
            data" for transmission to master
}
```

4.2.3 Configuration

Configuration of a node is XML based and makes use of the defined generic functional profiles. The UA developer references the desired functional profile(s) and datapoints using their id attributes and sets their values. Furthermore, group objects are created by associating datapoints

to group addresses. Besides, relevant parameters for the I/O library (e.g., mapping an output of the MCU to the corresponding light) can be set. An example device configuration is shown in Listing 4. This configuration file is being processed by a configuration tool, which transfers this representation into a suitable binary format and stores the information in the EEPROM of the target node.

Listing 4: Device configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<DeviceConfig xmlns:devconf="http://[...]">
        <ProgramID>1</ProgramID>
        <KNXLibrary>
                <IndividualAddr>1.1.1</IndividualAddr>
                <GroupAdresses>
                        <GroupAdress id="ga1">0/0/1</GroupAdress>
                        <GroupAdress id="ga2">0/0/2</GroupAdress>
                </GroupAdresses>
                <FunctionBlock id="417">
                        <GroupObjects>
                                <GroupObject id="go1">
                                        <DataPoint dpref="1" garef="ga1"/>
                                        <DataPoint dpref="2" garef="ga1"/>
                                </GroupObject>
                        </GroupObjects>
                        <Parameters>
                                <Parameter id="3">
                                        <Value>120</Value>
                                </Parameter>
                        </Parameters>
                </FunctionBlock>
        </KNXLibrary>
        <IOLibrary>
                <LED id="1" goref="go1">
                        <MCUPin>P70</MCUPin>
                </LED>
        </IOLibrary>
</DeviceConfig>
```

4.2.4 Development workflow

This section shortly describes the development workflow – starting from program development to final UA download – and a simple lighting UA.

The system software itself has to be compiled and downloaded using the MCU specific toolchain (e.g., avr-gcc and AVR Downloader/UploaDEr, avrdude). UAs may be developed and compiled using standard Java toolchains (e.g., Eclipse SDK and the SunJDK) and utilizing the User API. XML device configuration files currently have to be written by hand. The configuration tool (NanoVMTool) is being used to prepare the compiled class files, which includes stripping unnecessary and unsupported instructions from the binary as well as mapping native Java library calls to their corresponding implementations in C. Besides, the NanoVMTool has been extended to provide functionality for parsing the XML based configuration file and chang-

ing application parameters. Finally it is used to upload the UA into the sandbox via the serial interface.

Listing 5 shows the required UA to control a light (KNX functional profile 417). A node configuration defining the required group object and parameters is assumed (like in Listing 4). First the $KNXGroupObject$ has to be declared and instantiated using the $go1$ parameter. The main application code is then placed in an infinite loop. Here, first group object $go1$ is checked for changes which could be present if a related KNX message has been received. If there are any, the datapoint type value is read and simply used to set the LED accordingly.

Listing 5: Simple lighting application

```
package examples.sebas;

import nanovm.IOLib.io;
import nanovm.KNXLib.*;

class GroupObjects {
  KNXGroupObject go1=new KNXGroupObject("go1"); // configuration is
      automatically read using the id "go1"

  public static void main(String[] args) {
    System.out.println("Group objects example");

    while(true) {
      if (go1.hasChanged()) {
        DataPointType dp = go1.readValue();
        io.led(1,dp.intValue());
      }
    }
  }
}
```

5 Evaluation

To evaluate and test the stability of the stripboard using the Freebus circuit as well as KNXcalibur, a test environment was established. It consists of both boards, three pushbuttons and two lamps all connected to the same KNX line. Two of the pushbuttons were configured to toggle switch the lamps. A desktop computer was also attached to allow comfortable bus monitoring using the Engineering Tool Software (ETS). Figure 7 depicts the test environment. The boards have been used in various test cases, for example to switch one of the lamps, listening to the pushbutton commands or combined interactions like switching a lamp when a command of a pushbutton was received.

The presented approach clearly imposes an overhead on performance as well as on memory consumption. Therefore investigations of the SAC stripboard have been performed [14]. The code size is about 8 KBytes thus leaving another 8 KBytes flash memory for further extensions. The internal 512 Bytes and external 2 KBytes EEPROM remain free and are used to store the UA and configuration parameters. 326 Bytes SRAM are used for the operation of the sandbox, which leaves 580 Bytes for the UA. For a first performance evaluation, we implemented

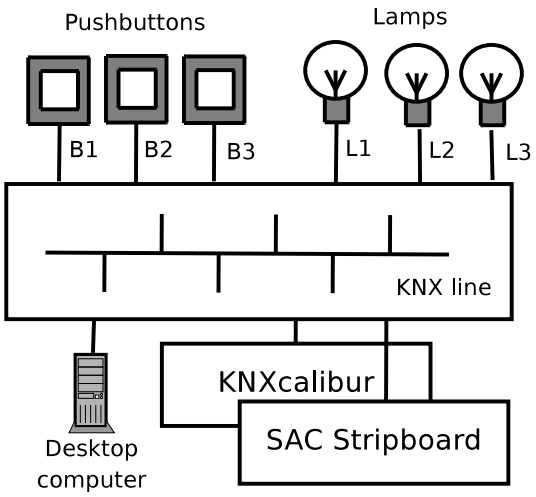


Figure 7: The test environment

Matrix dimension	Native C	Java int. EEPROM	Java ext. EEPROM
2	31 μ s	13.2 ms	420 ms
4	440 μ s	20 ms	2.62 s
8	3.92 ms	595 ms	18.5 s
16	32.8 ms	4.64 s	/

Table 1: Runtime matrix multiplication on SAC stripboard

the simple UA mentioned before as well as a simple PID controller. No subjective feeling of slowdown can be observed compared to standard C code. Since SACs in BAS typically do not perform more computationally expensive tasks and our used stripboard represents the lowest end available hardware we consider our approach working. To objectively estimate the performance impact of the sandbox approach we measured the raw calculation capability of square matrix multiplications with different matrix dimensions. It has to be noted, that no optimizations have been performed. Implementations in C and Java, running in the NanoVM using either internal or external EEPROM as program storage, are compared (cf. Table 1). The measured durations of the native C implementation are always three to four orders of a magnitude smaller than the corresponding Java versions using the external EEPROM. Memory access operations due to slow connection of this EEPROM have been identified as a major reason for slowdown. Thus we plan to replace the external EEPROM with a faster one. For performance analysis we moved the UA to the internal EEPROM. As can be seen this clearly improves the execution time.

6 Conclusion and future work

Despite the imposed overhead our presented approach allows development and execution of KNX UAs. Familiarization with KNX is being eased due to the help of a clean user API and the use of standard Java language, thus allowing a rapid development. Nevertheless, we plan to extend the current capabilities, most important a configuration tool to allow easy creation of

XML based configuration files. A possible ETS export of configuration data is being investigated just like the use of a generic BAS ontology. Besides, we plan to extend the current sandbox with features allowing more advanced programming methods (e.g., threads, exceptions) found in other virtual machines (e.g., TinyVM, JControl).

7 Acknowledgement

The authors wish to thank Thomas Flanitzer and Florian Guggenberger not only for their useful hints, but also for many helpful discussions and support in implementation issues.

References

[1] "KNX Specification", Konnex Association, 2004.
[2] W. Kastner, G. Neugschwandtner, and M. Kögler, "An open approach to EIB/KNX software development", in *Proc. 6th IFAC Intl. Conference on Fieldbus Systems and their Applications (FeT '05)*, Nov. 2005, pp. 255–262 (preprints volume).
[3] A. Kinne, "A new family of bus interface modules for KNX TP1", in *Proc. Konnex Scientific Conference*, November 2007.
[4] Siemens, "Technical Data EIB-TP-UART-IC", 2001, version D.
[5] W. Kastner and C. Troger, "Interfacing with the EIB/KNX: A RTLinux Device Driver for the TPUART", in *Proc. 5th IFAC Intl. Conference on Fieldbus Systems and their Applications (FeT '03)*, July 2003.
[6] B. Malinowsky, G. Neugschwandtner, and W. Kastner, "Calimero: Next Generation", in *Proc. Konnex Scientific Conference*, November 2007.
[7] A. S. Tanenbaum and M. van Steen, *Distributed Systems: Principles and Paradigms*, Prentice Hall, 2002.
[8] *Java Platform Micro Edition website*, http://java.sun.com/javame.
[9] T. Harbaum, *The NanoVM - Java for the AVR*, http://www.harbaum.org/till/nanovm/index.shtml.
[10] J. H. Solorzano, *TinyVM- Java for LEGO Mindstorms*, http://tinyvm.sourceforge.net/.
[11] T. Flanitzer, "Security mechanisms for low-end embedded systems - A Proof-of-Concept for Home and Building Automation", Master's thesis, Vienna University of Technology, Institute of Computer Aided Automation, Automation Systems Group, 2008.
[12] *FreeBus website*, http://www.freebus.org/.
[13] F. Praus, W. Kastner, and G. Neugschwandtner, "A versatile networked embedded platform for KNX/EIB", in *Proc. Konnex Scientific Conference*, November 2006.
[14] F. Praus, T. Flanitzer, and W. Kastner, "Secure and customizable software applications in embedded networks", in *Proc. IEEE International Conference on Emerging Technologies and Factory Automation ETFA 2008*, 2008, pp. 1473–1480.

Towards a Human Centered Infrastructure for KNX enabled Intelligent Environments

Christian Kleine-Cosack, Thomas Plötz and Gernot A. Fink

Intelligent Systems Group, Robotics Research Institute,
Technische Universität Dortmund, Dortmund, Germany
{Christian.Kleine-Cosack,Thomas.Ploetz,Gernot.Fink}@udo.edu

Abstract. Recent projects in the field of Ambient Intelligence (AmI) concentrate on technical aspects to integrate evolving ubicomp techniques into so called smart rooms and smart houses. The concept of context awareness is utilized to create what is called intelligence in such scenarios and the representation of user information on system level in general takes place within a general context model. While technologically impressive, a lack of portability to practical applications can be observed for the majority of such research projects. The tremendous dependency on special and expensive hardware limits the scope of application and prevents Ambient Intelligence for the masses. Additionally the technology centered view accompanied by an (implicit) data driven and non-active user model carries along different usability concerns. As a result the user turns out to be a foreign object on system level, which is excluded from the system's communication layer and – ignoring the user's skills – is merely handled as an obstacle in a technology centered system.

In this paper we present a case study featuring a novel approach to the integration of humans into AmI environments to prepare the ground for a user centric – thus user friendly – infrastructure for smart devices. The key aspect of the concept which we call human centered AmI is a dynamic and active user model which creates a virtual doppelganger of the user on software level. Mapping the user's capabilities and skills to system level, human services integrate seamlessly into the KNX enabled intelligent environment, allowing for transparent human computer interaction. Moreover, it also widens the scope of existing hardware and software components. In particular, the functionality of KNX devices in practical scenarios is complemented by human services and the presented approach.

The concept of human centered AmI is put into effect within the perception-oriented intelligent environment FINCA. This smart conference room is based on standard hardware components and, on sensor level, focuses on visual and acoustic data. In consequence our concept is easily transferable between related scenarios putatively exhibiting different hardware settings.

1 Introduction

Today the availability of more and more cheap and powerful computing devices together with their substantially and continuously shrinking size allow for realization of ubiquitous and disappearing computing. Intelligent environments integrate such technologies to silently serve the user in everyday situations and create what is called *Ambient Intelligence* (AmI) [1]: A supporting, omnipresent environmental intelligence, centered around the user.

In the past years tremendous progress was made by a variety of research projects in the field of AmI, defining the vision of intelligent environmental functionality from the viewpoint of the user. As a prominent example so-called *Smart Houses* left the laboratory state and now allow for practical application in realistic scenarios. Various actuators and sensors are integrated into the private home using ubicomp techniques, aiming for different application areas, e.g., enhancing the living rooms or conference rooms . Key aspect in such scenarios is the realization of context awareness at a global scale.

Being technologically impressive the transfer of recent findings to consumer relevant, real life applications is obstructed by tremendous and cost-intensive hardware dependencies. A multitude of sensors and actuators must be deployed to provide intelligent functionality. We call this gap between environmental intelligence or complete dysfunction of the AmI environment the *all-or-nothing dilemma*. In consequence conventional concepts in the field of AmI prevent a piecewise build up of environment on hardware level but force a complete all-or-nothing hardware setup. However for market relevance and, in particular, for open environments and uncontrolled conditions, a technological basis capable of being extended by time is indispensable. The KNX technology [2] can build such a hardware basis but will not be able to bridge the conceptual gap for AmI applications for the masses. In order to solve obstructing dependencies and to provide intelligent Human Computer Interaction (HCI), intelligent functionality w.r.t. user orientation demands for a global concept to dynamically integrate and couple hardware based and software services.

In [3] we presented an approach for an infrastructure for Ambient Intelligence environments which is centered around the user. In this paper we focus on practical aspects of this new concept of human centered AmI for it's implementation in real-world scenarios, namely smart environments. Especially the role of KNX devices for carrying the proposed concept from vision to practical application is considered. It is this practical issue that allow for the realization of close-to-market solutions beyond laboratory prototypes.

The discussion of this paper is organized as follows. In section 2 the new concept of human centered AmI in contrast to traditional approaches is reviewed. Subsequently in section 3 the realization of human centered AmI within an practical oriented scenario is presented and the application of KNX technology to the presented concept is discussed afterwards. The paper closes with a summary.

2 The Concept of User in AmI

By definition the agile user in everyday situations is identified as the fundamental aspect of Ambient Intelligence and is served by numerous computer based systems, thereby improving his experience. However, the center of focus in the multitude of practical oriented projects in this field can be identified in managing a versatile hardware zoo which provides smart functionality. On technical level, exemplary, two prominent projects, the Gator Tech Smart House [4] and the inHaus Duisburg [5] realize Ambient Intelligence at a large scale. Within realistic domicile scenarios, a multitude of sensors and actuators is integrated to create an AmI environment solving diverse problems of hardware-integration. Certainly, these aspects of hardware integration prepare the ground for every AmI environment. Context awareness, as being the key aspect to differ smart automation processes from intelligent environmental behavior, is realized by a dynamic, integral context model. The user as part of the context as well as his relation to environmental objects is implicitly modeled within this passive and data based concept of context.

We now switch the viewpoint from the technical aspects, turning towards the characteristics of the user, his persona, abilities and skills. In fact the user is the integral part of every AmI environment, predominantly and actively determining the state of the context. In consequence, his characteristics must be first and foremost taken into account within the context modeling process. Conventional approaches in the field of AmI model the user within a passive, integral context model. In contrast in this approach the user is modeled separately allowing to take into account his special characteristics in the modeling process.

Moreover in practical oriented, complex AmI scenarios the user is confronted with ubiquitous systems he might not see, with intelligent functionality he might even not be aware of. This novel situation in Human Computer Interaction raises further usability concerns for AmI environments, beyond the technical scope of hardware integration.

2.1 Human Skills and a Proactive User Model

In our approach of human centered AmI which is briefly summarized here [1] the central component of the context model is an autonomous and active software agent representing the user on system level. In contrast to conventional approaches which implicitly model the user within a global and integral context model, this concept of a *virtual doppelganger* separates the user from an integral context model. However, the relationship between the overall environment context components is tight. The respective mutual interferences are illustrated in figure 1. But the separation of the user model enriches its potential, allowing to map the user's behavior and, moreover, the user's skills to the model, the realization of which we will call *human services*. In consequence the user and

[1] For a more thorough discussion of the concept of human centered AmI based on human services cf. [3]

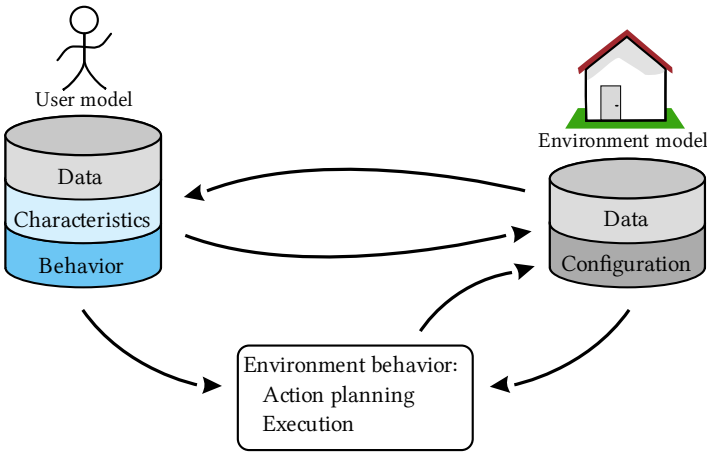


Fig. 1. There is a close relation between user and context model. The state of the environment directly influences the role of the user and, vice versa, the user implicitly determines the current state of the environment (taken from [3]).

his characteristics are integrated on system level the motivation of which is to overcome the aforementioned all-or-nothing dilemma and allow for system wide intelligent, adaptive and user-guiding HCI.

The concept and the realization of human services is inspired by the the UPnP technology [6] which is the de facto standard for service discovery, description and control in the field of consumer electronics. Numerous hardware devices and software applications implement the defined interfaces . Software agents and gateways allow to refit this technology to existing but innately incompatible devices to dynamically deploy functionality. Similarly, the human user brings in a variety of high level skills into the AmI environment but, up to this point, a solution for integration of those is not existent. The concept of human services proposes a *human control point*, offering his capabilities as software services. This allows for dynamic coupling of off-the-shelf hardware devices, based on KNX [2], UPnP [6] or OSGi [7] technology, just to mention some examples of putative underlying technologies.

On level of function, human services can be divided into two categories. First, communication services implement HCI interfaces which, while dynamically adopting to the user, realize a system wide intelligent communication channel between user and his environment. In consequence messages to the user are delivered dependent on available output devices, selecting appropriate modalities and language. As a prominent example a text message can be read to the (blind) user, visually presented or even suppressed if the user gives a talk and the message is of minor urgency. The second category of human services provides the user's skills as environmental functionality, making use of communication services itself. Exemplary a door-open-service is offered by the user which can be coupled with conventional services, e.g., controlling light conditions. This ser-

vice can be demanded by different software or hardware components of the AmI environment.

Following an object oriented approach, decisions on service offering and withdrawal are taken by the proactive user model, depending on knowledge about the user, his current activity and global context information.

2.2 Human Centered AmI

The dynamic and active user model, utilizing the concept of human services, enhances conventional approaches in the field of AmI. The user is integrated into environment on system level, taking the central role within the overall context model. Hence, we call this concept Human Centered AmI offers a way out of the proposed all-or-nothing dilemma and enables intelligent HCI. The different aspects of the concept – the context model's components, the virtual doppelganger, the integration of human services in strategy planning – are illustrated in figure 2.

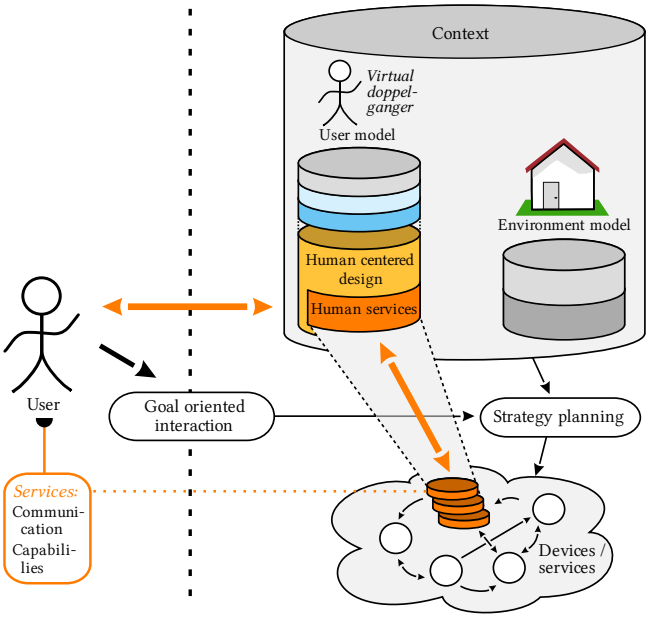


Fig. 2. Human services allow for the integration of the user's capabilities into the strategy planning process of intelligent environments and are the key feature of Human Centered AmI. On the left hand side the real user is shown, on the right his virtual doppelganger. The integration of the user model into the system is accomplished at service level. The role of the user shifts from being the trigger of system activities to an active component within the overall system. The user model thereby controls the intelligent communication between the user and his virtual doppelganger (adopted from [3]).

Fig. 3. Overview of the FINCA (left) and inside view of the smart conference room.

This process of user integration follows the paradigm of service orientation. Hence, an implementation of our approach using standard service oriented software frameworks is feasible. We propose this concept for building smart, extensible and user centric spaces on basis of standard technologies.

3 From Vision to Application

Directed towards the user, the focus of Ambient Intelligence lies on usability in general, context awareness and HCI to be major aspects of interest in this field. In concordance, we proposed the concept of Human Centered AmI. Furthermore AmI clearly defines the target audience – the (general) mobile user in everyday situations – which sets tight constraints w.r.t. hardware deployment. Hence, for practical relevance of research findings, evaluation and discussion has to be conducted w.r.t. this background. In the following we will present an AmI environment, designed to meet these constraints. We will refer to a case study on usability in this environment, analyzing the concept of Human Centered AmI to, finally, return to the hardware level discussing aspects relevance in practice.

3.1 The FINCA: A Perception Oriented Intelligent Environment

The FINCA, a Flexible, Intelligent eNvironment with Computational Augmentation [8] provides the context for the work presented in this paper. Aiming at the development of techniques for sophisticated and natural HCI, sensor data related to human perception are processed using statistical pattern recognition techniques. In this project a Smart House has been build, consisting of basically two areas: A smart conference room equipped with numerous sensors (e.g. cameras, microphones and light switches) and actuators (e.g. light and sun-blind control units). Furthermore a lab-space exists, serving different scientific research projects. An overview of the FINCA building and the conference room is shown in figure 3.

To serve as a practical oriented scenario for AmI applications and to assure transferability of findings to different (real life) scenarios, only standard, off-the-shelf hardware components have been chosen for the set-up of environment. In

particular KNX technology [2], providing the standard for home and building control, is used for numerous tasks within the FINCA. Furthermore the OSGi technology [7] serves as service oriented framework for the FINCA. On top, numerous software applications implement gateways between different communication systems and transparently provide unified software services across technologies. In example, only through services provided by the OSGi framework, a 3D-gesture-detection service, running on a remote computer, demands and receives camera images via network and accordingly controls light units which feature the KNX technology.

Being a platform for high level applications the FINCA is a cooperative house environment supporting users during various activities. Detecting, locating and tracking its communication partners by analysis of visual and acoustic data, the FINCA allows for multimedia scene analysis and natural HCI. In consequence it builds an adequate platform for human centered AmI.

3.2 The Role of Home and Building Control for Successful AmI

So far it has been shown that human centered AmI, in principle, offers a solution for user integration for next generation smart environments. In fact the human centered AmI represents a step beyond existing concepts for user and context modeling in AmI environments. Taking into account the specific characteristics of a human user, the virtual doppelganger and human services potentially allow for high level Ambient Intelligence.

One goal of Ambient Intelligence is to develop general solutions for everyday situations, i.e. solutions that work in practice. Thus technical aspects of concrete realization have to be considered to assure for market relevance. Analyzing typical use-cases of smart environments it becomes obvious, that smartness is directly linked to natural and intuitive, thus intelligent, human computer interaction tasks. Moreover high responsiveness of an environment is a basic necessity for this to achieve. A human user interacting with the intelligent house expects appropriate reactions of the technical system. For example, controlling of light conditions via high level services, e.g., speech and gesture recognition, in practice demands for immediate environmental activity – the light shall actualy be turned on or off.

To allow for the necessary physical reaction at the technical level a complex interaction problem w.r.t. concrete physical devices has to be solved. At this point technologies for smart home and building control come into play by linking the field of electrotechnical installations to AmI. In concrete, the well established KNX technology offers the desired functions for smart actions and automation processes within intelligent environments. Numerous hardware devices, i.e. sensors, actuators and control units, can be explicitly addressed by the environments software framework and dynamically coupled to miscellaneous services.

Different use cases illustrate the flawless collaboration of approved home and building control technologies and the concept of human centered AmI. To pick up

the previously mentioned example, controlling lighting conditions can be accomplished by standard switches or, more sophisticated, by multi-functional-control panels, PDAs or smartphones and so forth. But integrated into the concept of human centered AmI the actuators can be controlled by intuitive interaction means, namely using natural modalities like speech and gestures. Even personal preferences of the user can be taken into account for the controlling task. The KNX technology introduces an layer for abstraction, allowing for "soft-wiring" of devices and, in consequence, dynamic coupling w.r.t. intelligent environmental activity. In contrast to hard-wired logical entities of devices, this message based infrastructure bridges the gap between the desired natural interaction and it's grounding to physical actions.

Furthermore, every newly integrated KNX device not only enlarges the functional pool but, coupled to human services can be lifted to new application areas. For example, the actual function of a conventional switch can depend on a pointing-gesture of the user. In consequence the KNX technology can draw benefit from high level AmI functionality and in turn can manifest it's position within the infrastructure of environment. Integration of conventional hardware devices – devices, first and foremost not designed for intuitive HCI – on service level allow for high-level Ambient Intelligence, beyond the domain of home automation applications.

To sum up, it is the concrete installation of electric devices and their integration into a framework for home and building control that represents the basis for responsiveness in AmI environments. Consequently, the KNX infrastructure is, to some extent, the pre-requisite for "AmI for the masses".

4 Summary

One goal of Ambient Intelligence is to develop concrete applications that support the users in their everyday life in an intuitive and natural way. Along this way cost-efficiency and broad availability of hardware are considered as constraints. However to reach the mass-market, AmI approaches must solve the all-or-nothing dilemma, which we identified for conventional AmI applications: If an application relies on specific software and hardware services the overall application is hardly transferable, even to related scenarios. For example, AmI solutions developed for Smart Houses substantially depend on the specific hardware setup and will fail otherwise. Furthermore usability issues arising from the fact that the user is surrounded by a multitude of ubiquitous computing systems need to be solved.

In our work we developed an approach which focuses on the user and, respectively, the user's model on system level. This concept we call human centered Ambient Intelligence. A virtual doppelganger is created which represents the user, his characteristics and behaviour on system level and actively co-determines activities within the software framework. In consequence, the user is transparently integrated into the Service Oriented Architecture, thereby offering his skills as so called human services. A case study on usability showed the effectiveness

and acceptance of this concept in practice. Moreover it was argued that the all-or-nothing dilemma, putatively, can be resolved by this concept.

The discussion on home and building control technologies for AmI environments showed the mutual benefit w.r.t. practical application. To be precise, the KNX technology proved to provide a solid infrastructure to realize responsiveness within AmI environments. In return, conventional KNX devices can draw benefit when coupled to other high level services. Consequently the KNX technology can play an important role for Ambient Intelligence in real-world settings.

References

1. IST Advisory Group (ISTAG): Ambient intelligence: from vision to reality. Technical report, Brussels, Belgium (2003) ISTAG draft consolidated report.
2. KNX: Knx - the world's only open standard for home and building control. http://www.knx.org/
3. Plötz, T., Kleine-Cosack, C., Fink, G.A.: Towards Human Centered Ambient Intelligence. In: AmI-08: European Conference on Ambient Intelligence, Darmstadt, Deutschland, Springer (2008) to appear.
4. GatorTech: The Gator Tech Smart House. http://www.icta.ufl.edu/gt.htm
5. inHaus: Das inHaus-Innovationszentrum der Fraunhofer-Gesellschaft. http://www.inhaus-zentrum.de
6. UPnP: The UPnP Forum. http://www.upnp.org/
7. OSGi: OSGi - The Dynamic Module System for Java
8. Plötz, T.: The FINCA: A Flexible, Intelligent eNvironment with Computational Augmentation. http://finca.irf.de (2007)

Enhancing residential automation systems with artificial intelligence

T. Abinger[1] W. Kastner[1] G. Luber[2] G. Neugschwandtner[1]

[1]Automation Systems Group
Vienna University of Technology
{tabinger, k, gn} @ auto.tuwien.ac.at

[2]Industry Sector, BT ET Q
Siemens AG
georg.luber@siemens.com

Capable home automation systems represent a considerable investment. To be able to offer their users increased value in return and fully use the potential of instrumenting residential environments with sensors and actuators, controllers need to become more intelligent. Only then they will be able to create environments tailored to the individual wishes of every tenant while still saving energy cost. Not only is this a highly complex task that needs to be structured properly; it necessarily includes that controllers can autonomously adapt to the circumstances. En route to enhancing residential automation systems, our first goal is to provide autonomous adaptation of temperature setback schedules for independent rooms based on an artificial neural network controller. The controller is trained and tested with data originating from a building simulation environment. A comparison to a naive controller strategy is given. Next, the agent software engineering paradigm is introduced. Based on a methodology for engineering agent-oriented software, a society of agents is presented. Beyond others, User Agents hold information about occupants' preferences and habits, acting on their behalf to ensure these are met, and a Control Agent determines and enforces optimized lead times (optimum start/stop control).

1 Introduction

The application of automation technology to residential environments holds a lot of benefits. Still, much of the potential available in a typical present-day home automation system lies fallow since the control strategies linking sensors and actuators are not as flexible as they should be. Tuning such a system precisely to the requirements of its users and the characteristics of both building structure and building services equipment is a task reserved to those with specialist knowledge. Moreover, it is almost never done in full due to the large effort required. For the same reason, once the system is installed, necessary re-adjustments are foregone almost as a rule. The task gets even harder as more design disciplines are involved.

Only with control getting more intelligent, comfort and energy savings can be maximized and investments in sensors and actuators repaid in full. Imagine a home that would be like this: In the morning, the alarm rings to wake Bob. Even before it has rung, the temperature of the house was brought from power-saving to comfort level just in time – although the wake-up time set by Bob was different from the day before. After getting up, it is time for a shower. Again, the boiler was heated in advance to provide a sufficient amount of hot water. After getting ready for the day, Bob enjoys breakfast with fresh coffee – the espresso machine had of course heated up on time. When Bob enters his car to drive to work, the door of the garage opens automatically. After he has left, it automatically closes again, the alarm system is enabled and room temperatures are brought down to an energy saving level.

For such a scenario to become true, an automation system has to go far beyond handling predefined control loops. Rather, it must be able to actually perceive what is happening in the home and take the proper actions to make its occupants comfortable. This includes selecting the most appropriate course of action leading to a particular goal and planning ahead to take expected future situations into account. For this purpose, it needs to draw on knowledge about the occupants' preferences and behavior (such as their typical daily schedules) as well as the building structure and its properties (e.g., the thermal inertia of a room or the entire house), including all building services (and their effect).

This knowledge may become obsolete over time. Obviously, occupant preferences change. However, if a large tree covering a south window is cut down, the room behind will also react differently to outside conditions all of a sudden. It will also be necessary to continually resolve goal conflicts: Bob's wife may find a higher room temperature comfortable than he does. In terms of energy consumption, not heating or cooling at all would give an optimum result. What is considered the best compromise strongly depends on individual preferences, which the system has to be aware of.

An automation system cannot be feasibly provided with this information in the way today's systems are configured. Although, for example, it could be preloaded with an understanding of the laws of physics, the entire wealth of knowledge described quite obviously cannot be engineered manually. Therefore, systems must be able to acquire this project specific context autonomously. Future home automation controllers must be enabled to learn from any user – not only qualified personnel and experienced users – and, as much as possible, by themselves from what they perceive about their environment.

All the tasks described are classic examples of artificial intelligence (AI). This paper intends to remove a bit of the mystery that may still shroud AI tools and approaches. It is divided into two parts – one taking a bottom-up approach, the other going top-down. First, it is shown how a standard present-day heating system could be enhanced by applying AI. An artificial neural network is used to predict heating lead times from historic data without manual intervention. It is also shown how such a control strategy can be evaluated using building simulation. The second part introduces the agent concept. However, its focus is less on intelligent agents as an AI paradigm than on agent-oriented software engineering. Agents as a software engineering concept provide a convenient and powerful way of structuring a complex software system where entities operate with a considerable degree of autonomy. An agent system architecture that covers all the tasks outlined above is presented. Implementation considerations (such as suitable tools and standards) are also covered.

2 Using AI for optimum start and stop

It is an obvious fact that the temperature in a room does not rise the moment one opens the radiator valve. Rather, the thermal inertia of walls and furniture causes a noticeable delay. Thus, heating (and cooling) systems need to be activated a certain time before a room or zone is expected to provide comfortable conditions. On the other hand, it may be possible to deactivate them already some time before occupants are expected to leave, since the temperature will not leave the comfort zone instantly. In building automation, this is known as optimum start/stop operation.

The start/stop time offsets can of course be determined manually. However, this requires some discipline. Moreover, these offsets change with the season. Thus, they will likely be chosen too long when starting and too short when stopping, just to be on the comfortable side. Therefore, it would be preferable if the system would be self-learning. The thermal performance of the building should be detected automatically, and lead times adapted accordingly to satisfy comfort requirements with a minimum of energy required.

In order not to let the problem get out of hand, we shall assume that occupant presence follows a strict schedule that is known in advance. The task, therefore, is predicting how long in advance heating (or cooling) systems have to be started (and can be stopped) so that occupants still find comfortable conditions while they are present. These predictions shall be made from historic data accumulated during normal system operation.

2.1 Field level interface

Ideally, we would like our prediction to take everything into account the lead time depends on: room cubature, building materials, furniture, surface temperatures, core temperatures, inside and outside air temperatures, additional loads, convector or radiator area, flow temperature, ... However, if the system is to be applied in a realistic, typical home setting, few of these parameters will be available. Only some will be known, and even fewer will be available via a suitable data interface.

Thus, it shall only be assumed that the current room or zone air temperature – as measured by a typical wall sensor – and its set temperature are available. We shall also assume that the current outside air temperature is available. In case no sensor is present, the latter can be obtained via Web services. Any other actual values will have to be estimated by drawing conclusions from these observations based on domain knowledge.

Once the latencies have been determined, the necessary adjustments need to be made to the heating/cooling schedule. This is straightforward if the occupant presence schedule is given. However, adjusting the schedule by a static amount fails to take into account that the time offsets may depend on outside conditions that cannot easily be predicted.

Therefore, it shall be assumed that our system can modify the setpoint of the heating/cooling system controller. Besides being a reasonably realistic assumption given the interfaces typically offered by such systems (especially if only toggling between an “economy” and a “comfort” set temperature is required), this approach also allows a generic solution that is independent of the type of heating or cooling system in use.

2.2 Neural network implementation

Artificial Neural Networks (ANNs) are modeled after the vast neural network of human brains. An ANN consists of one or more neurons which are linked among each other. A single neuron consists of multiple inputs, accepting stimuli, and one output. If the weighted sum of the stimuli exceeds a certain threshold, the neuron fires, creating an output stimulus. Typical ANNs consist of an input layer with one neuron per input, an output layer with one neuron per output, and one or more "hidden" layers in between which store the internal knowledge used for processing the output. The links between neurons are directed and weighted. In general, the global direction of the edges is directed from the input to the output layer, but loops from one layer to layers before are also possible.

The type of our ANN (Figure 1) is a multi layer perceptron, also called competitive network. It has a feed forward architecture: information moves in one direction, from the input nodes through the hidden nodes to the output node. There are no cycles or loops in the network. The ANN has 3 inputs:

- Current room air temperature
- Current outside air temperature
- Room air temperature to be reached

Its output is the time it will take to reach the desired room temperature.

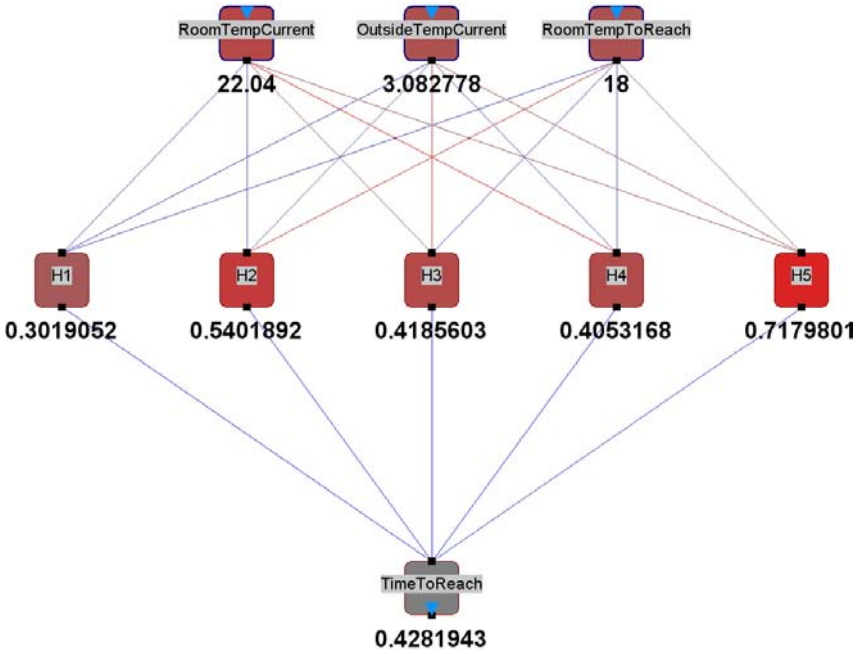


Figure 1: Artificial neural network

For most ANNs only one hidden layer is sufficient, which consists of at least as many neurons as the input layer, but not more than twice as many. For this reason, we have selected one hidden layer with 5 neurons.

Our ANN is trained by applying multiple sets of input and output values that exhibit the desired relationship between inputs and outputs (supervised learning). During these sessions, the link weights between the artificial neurons are adapted. We used an adapted version of the backpropagation learning algorithm, the so called resilient backpropagation learning algorithm. This is a common heuristic algorithm for supervised learning. The workflow for supervised learning can be described as follows:

1. Set the input pattern on the input neurons.
2. The input propagates through the net in the forward direction and generates an output pattern on the output neuron.
3. The current and the correct output values are compared, and the difference between the two values is calculated.
4. The error between the set value and current value is calculated using an error function. Then the result is backpropagated from the output layer to the input layer, and the weights of the edges are recalculated to minimize the net error.
5. The weights on the edges are adapted as calculated in the previous step.

2.3 Evaluation

For training and optimization of our ANN, a data set from given input values to some future output values was needed. Thus, the challenge was to find an adequate environment to calculate training data first, and, subsequently validate the trained ANN. There are two options to generate training data and validate the test results:

1. Use real life data. All variables accurately reflect the real world. However, data can only be acquired in real time, heating and cooling take a (very) long time (thus not applicable in our case).
2. Use a simulation environment. Simulations can run much faster than real time. This allows testing variations while still being cost effective.

We started our efforts with the building simulation software EnergyPlus [1]. To set up the model and its parameters we used the commercial graphical user interface DesignBuilder, which has a built in EnergyPlus engine [2]. EnergyPlus is an excellent software for evaluating energy flows in a model. However, we found no way to change parameters within a running simulation.

In our second approach we evaluated the free library ATplus [3, 4]. The library is based on Modelica, a general purpose, object oriented modeling language for complex systems. Modelica is used in multiple domains, especially in physics and electronics. Modelica models are evaluated using simulation software such as Dymola [5] that is able to symbolically handle complex differential equation systems. ATplus provides dynamic thermal models of building structure elements (e.g., walls, air, windows) including heat flow and storage, weather models (weather related influences, solar radiation), HVAC components and controllers for heating (discrete and continuous controller models, but also fuzzy controllers). Moreover, it has been validated against TRNSYS [6], often referred to as the “gold standard” in building simulation.

2.3.1 Building model

ATplus comes with a simplified example (Figure 2) that we modified slightly. To handle the parameters some basic knowledge of building physics is necessary.

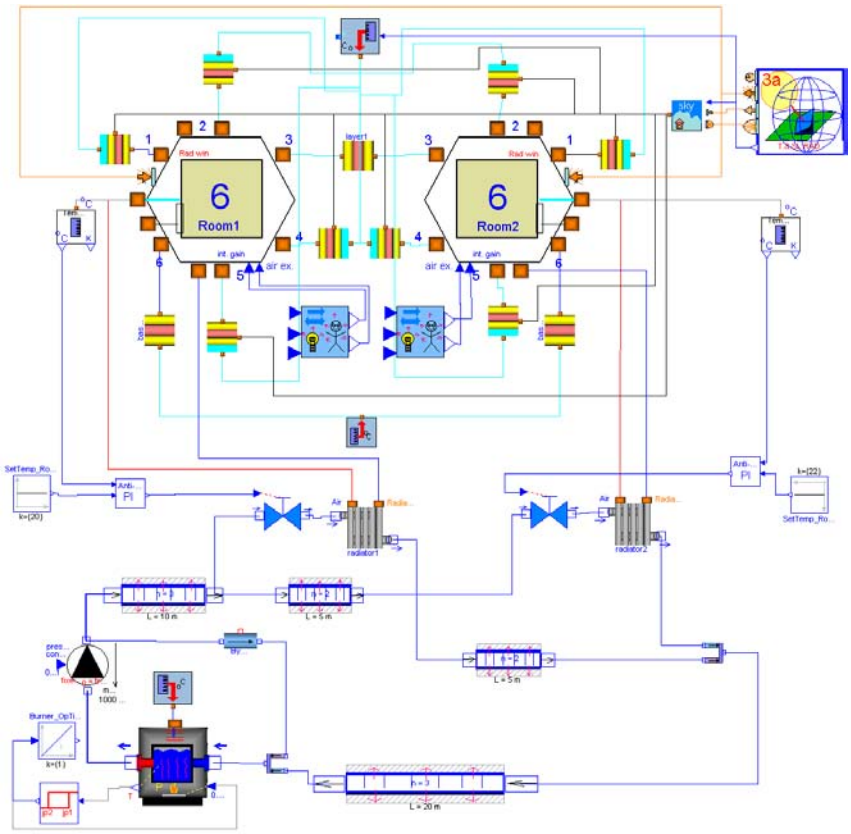


Figure 2: Simulation environment

Our building model has two rooms, each with a base area of 30 m^2 and a ceiling height of 2.80 m. Both rooms have a 40 cm aerated concrete base and are separated by 30 cm aerated concrete walls. The window area in each room is 6 m^2. Both rooms have a hot water radiator with 11.4 m^2 surface area installed. The water is heated in a central 200 l boiler with a 10 kW heat source. The flow temperature is 80 ^oC (corresponding to an old style installation). Both rooms are controlled by independent PI controllers. We defined the comfort temperature set point as 22 ^oC and the economy temperature set point as 18 ^oC.

The simulation was run for successive 15 minute periods of simulated time; after each period, actual values were read out and set points modified if required. Simulating one week took roughly 45 minutes on a Pentium 4 Dual Core with 2.13 GHz and 2 GB of RAM. The weather data used were logged in southwest Germany in March 2003.

2.3.2 Occupant presence schedule

The occupant presence schedule follows the typical daily routine of a family. The scenario is based on the idea, that parents and children leave home together on working days, but stay at home on weekends. For simplicity, we assume the two rooms of the simulation environment to

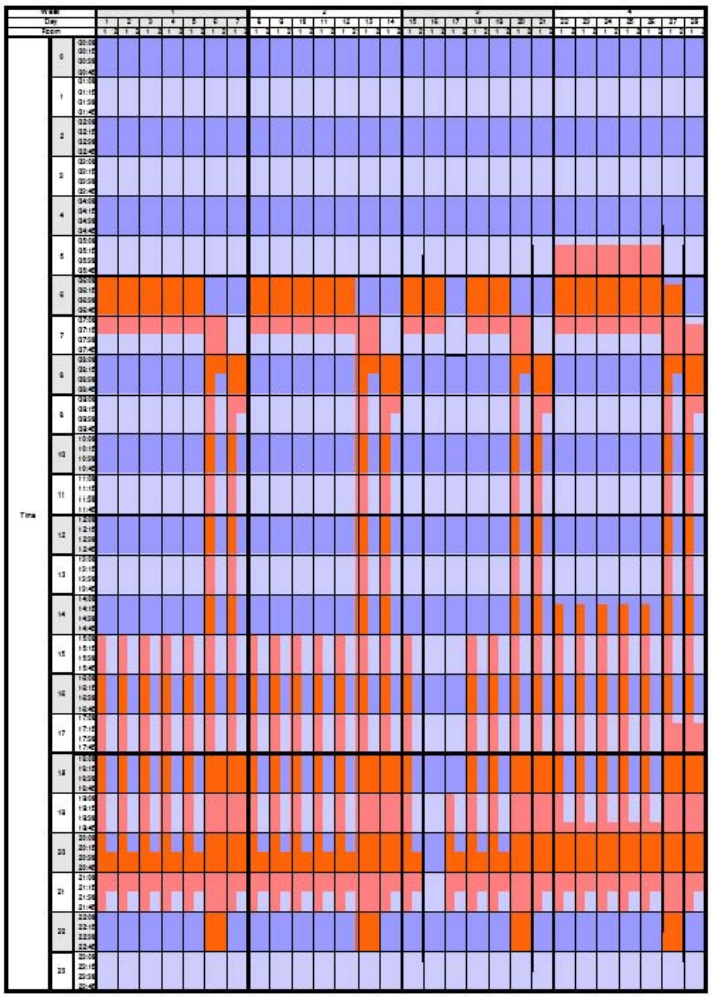


Figure 3: Schedule

be placeholder for a kitchen and a living room (room 1), and bedrooms and bathroom of the house (room 2).

The occupant presence schedule was created for one month operation. It is identical for each of the four weeks except for the third week where the family is for two days on vacation. From Monday to Friday both rooms are allocated from 6:00 till 7:30. This is the time when the family wakes up, showers, eats breakfast and finally leaves the house. In the afternoon at 15:00 the children get back from school. In the evening around 18:00 both parents return home. At 20:30 the bedrooms will be heated. During the weekend the kitchen and living room are heated all over the day, starting from 7:00 to 23:00. On Saturday, the bedrooms and bathroom are heated later in the morning. In the evening they are heated for a longer period starting from 18:00 until 23:00. The schedule for Sunday is similar, but starts one hour later in the morning and ends one hour earlier in the evening.

In the first week, the simulation starts from an initial state and all internal masses have to be heated up. This week is discarded and not used for later simulation and evaluation steps. From the second to the third week regular activity is done. We use the fourth week for validation purposes. Figure 3 gives an overview on the occupant presence schedule.

2.3.3 Simulation results

We compared two heating strategies by benchmarking the burner operating time, which corresponds with the energy used for heating. The first strategy is a very simple one: it just starts the burner 45 minutes earlier before the rooms are going to be occupied. This guarantees that rooms have reached their comfort temperature timely.

As a second strategy we use two ANNs as presented in Section 2.2, one trained for each room with data from weeks two and three. For each room, the ANN controller decides with regard to the occupant presence schedule when the room must be pre-heated. Every 15 minutes (of simulation time), it takes a look at the schedule and asks the ANN for a lead time estimate based on the current room temperature, the current outside air temperature and the desired room temperature. If the time estimate (rounded up to multiples of 15 minutes) equals the time till the next scheduled occupant presence, the set temperature will be changed to the comfort value. At the end of the presence period, the set temperature is lowered again. Figure 4 summarizes the software components involved.

Over one week, the naive strategy resulted in a burner operating time of 46.1 hours. With the ANN controller, the amount could be reduced to 44.8 hours. This corresponds to an improvement of 2.8 %.

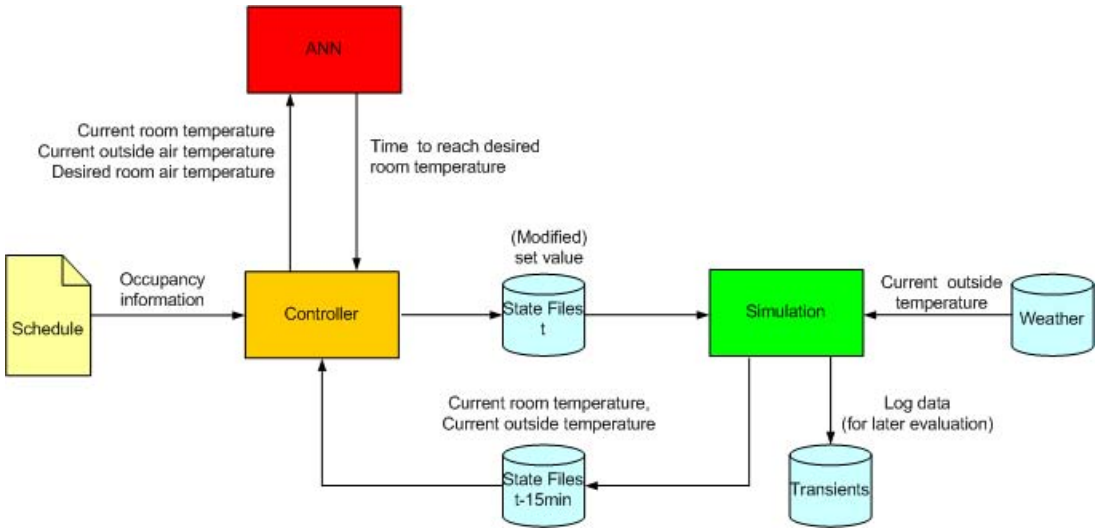


Figure 4: Test environment for ANN controller

3 Engineering intelligent systems

The ultimate goal of our project is to apply artificial intelligence not only to the optimum start/stop problem. Presence and location of tenants have to be detected, activities identified and preferences of the persons involved to be taken into account. In addition, the system has to be able to learn about the tenants' behavior and establish desired environmental conditions. In case of multiple persons present, it must also be able to resolve conflicts. The overall system needs extensive knowledge about the static structure and current and future behavior of the building. This knowledge could be combined into a common information model. All together,

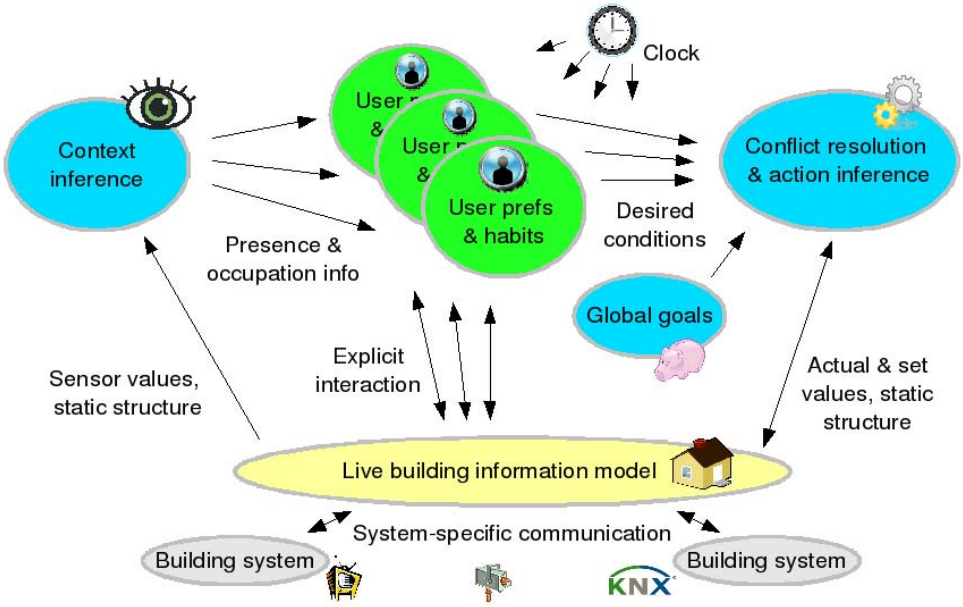


Figure 5: Tasks and information flow

an approach based on the software agent paradigm seems reasonable. Figure 5 summarizes the tasks and information flow of our system.

3.1 Agent based systems

From an AI perspective, intelligent agents continuously perform three functions: perceiving dynamic conditions of the environment, taking actions to affect conditions in the environment, and reasoning. Reasoning includes interpreting perceptions, solving problems, drawing inferences, and, finally, determining actions (cf. [7]). A society of agents is called an agent based system. Agents are operating continuously and can decide independent from other agents. However, they cooperate with each other to solve problems. Every agent has knowledge about the context in which an action takes place. [8] gives further classification criteria for software agents:

- Static or mobile: A static agent resides on a fixed host, whereas a mobile agent can move around a network from one host to another.
- Deliberate or reactive: A deliberate agent has the capability of reasoning, planning, and scheduling, which allows it to negotiate with other agents in order to achieve coordination. A reactive agent does not have any internal symbolic model of its environment. It uses a stimulus type of behaviors to respond to the present state of the environment in which it is embedded.
- Collaborative, interface and smart: A collaborative agent can cooperate and communicate with other agents. A smart agent would have to learn as it reacts and/or interacts with the external environment. An interface agent acts as the interface between an agent society and a human.

3.2 Agent modeling

Modeling an agent based system requires appropriate modeling techniques and tool support. One commonly used methodology is called Prometheus [9]. Prometheus allows to develop multi-agent systems in three phases: system specification, high-level (architectural) design, and detailed design. During the system specification phase basic functionalities of the system are identified, including inputs (percepts), outputs (actions) and any important shared data sources. This phase is subdivided into an analysis overview used to develop the high level view of the system requirements, and the identification of scenarios, goals and sub-goals. At its end,different goals, percepts and actions can be grouped as roles. This helps in further modularizing the system. The architectural design phase focuses on the outputs (i.e., roles) from the previous phase to determine which agents the system will contain and how they will interact. Here, data coupling, agent-role grouping, and interaction between agents come into play and are concentrated in a system overview diagram. Finally, the detailed design phase looks at the internals of each agent and shows how it will accomplish its tasks within the overall system. Communication within Prometheus is modeled with the Agent Unified Modeling Language, an enhancement to classical UML sequence diagrams. The Prometheus methodology is supported by the Prometheus Design Tool (PDT) [10].

3.3 System overview

Following the Prometheus methodology, Figure 6 shows the resulting system overview diagram of our multi-agent system. Rectangles stand for agents which collaborate either via specific one-way messages, an interaction sequence (protocol) or via data storage. Data storage is used to store and grant beliefs of an agent. Percepts are inputs coming from the environment (in our case, human interaction, sensor changes or the passing of time). We assume that all agents have access to a globally available building information model and that percepts and messages carry the current date and time.

- The **Interface Agent** acts as interface between tenants, automation system, and our agent society. It fills the process image and sends messages to the agent system if a sensor change happened or some user interaction occurred.
- The **Trending Agent** is responsible for logging trend data.
- The **Presence Agent** detects whether a person enters a room and requests data from the User Agents (see below) to identify him/her. After successful identification it tracks the person and reports his/her location continuously to the responsible User Agent.
- The **User Agent** has knowledge about the appearance, preferences and habits of a particular occupant (its owner). Moreover, it can identify his/her activities to build a model of the current situation. If a situation or sequence of situations (scenario) has been detected, the User Agent causes the preferred environmental conditions of its owner to be established. A User Agent is able to learn new identification patterns, situations, scenarios and owner preferences. For this purpose, user feedback is taken into account. If a person cannot be identified (e.g., a guest), an anonymous User Agent is dispatched to cater for his/her needs until the guest leaves.
- The **Conflict Resolution Agent** resolves potentially occurring user desire conflicts.

- The **Building Analysis Agent** periodically evaluates the room behavior of each room in the house.
- The **Weather Forecast Agent** predicts the weather based on data from a local weather station or the Internet.
- The **Control Agent** determines the optimum set values for building services equipment based on the current and expected future set points for each room. Its decision can be based on simple control algorithms, fuzzy control or neural networks. Additional information about the room behavior and weather forecast data are considered.

3.4 Implementation aspects

For the proof-of-concept implementation the open source software Jadex [11] will be used. It allows for programming intelligent software agents in XML and Java and can be deployed on different kinds of middleware such as JADE [12]. Jadex follows the Believe/Desire/Intention (BDI) model. Beliefs describe the knowledge about the world of an agent. Desires specify activities or goals which an agent wants to achieve. Intentions reflect what an agent would like to do. The agent development platform implements the FIPA Agent Communication Language ACL standards [13]. This standard describes a protocol for intra- and inter agent communication. In Jadex, data are specified with the help of the Resource Description Framework RDF [14]. Together with the Web Ontology Language OWL [15], RDF will provide the basic ingredients for the description of an ontology necessary for setting up our building information model.

4 Conclusion and outlook

The paper presented an approach enhancing residential automation systems with artificial neural networks. To evaluate the concept, the simulation environment Dymola was used together with the free and open source library ATplus. This library comes with a building model consisting of two rooms where basic parameters were modified. Sometimes, for some reason (e.g., when modifying the set point temperature), the simulation stops with an error message reporting that differential equations could not be resolved. The adjustment of other parameters such as internal loads (occupants, appliances) and internal masses (furniture) has not be considered so far. This might also be one reason why our current building model allows heating the rooms almost independent from the outside air temperature within 8 minutes, and cooling them from comfort temperature to economy temperature within 15 minutes. For properly setting the remaining parameters of the model, deeper knowledge in construction engineering and building physics would be beneficial – even more so since the type of construction greatly influences how and to which amount savings can be realized.

The developed ANN follows a feed forward architecture and takes 3 inputs only. To get better predictions from the ANN, we are considering the use of historical data (which allows to draw conclusions on values that cannot be measured directly, such as wall core temperatures). This also means that the ANN has to be redesigned and tested again, since choosing a suitable network type and layout is not trivial. Nevertheless, the present savings using our ANN controller are measurable, even when we take into account that simulations were done for one month

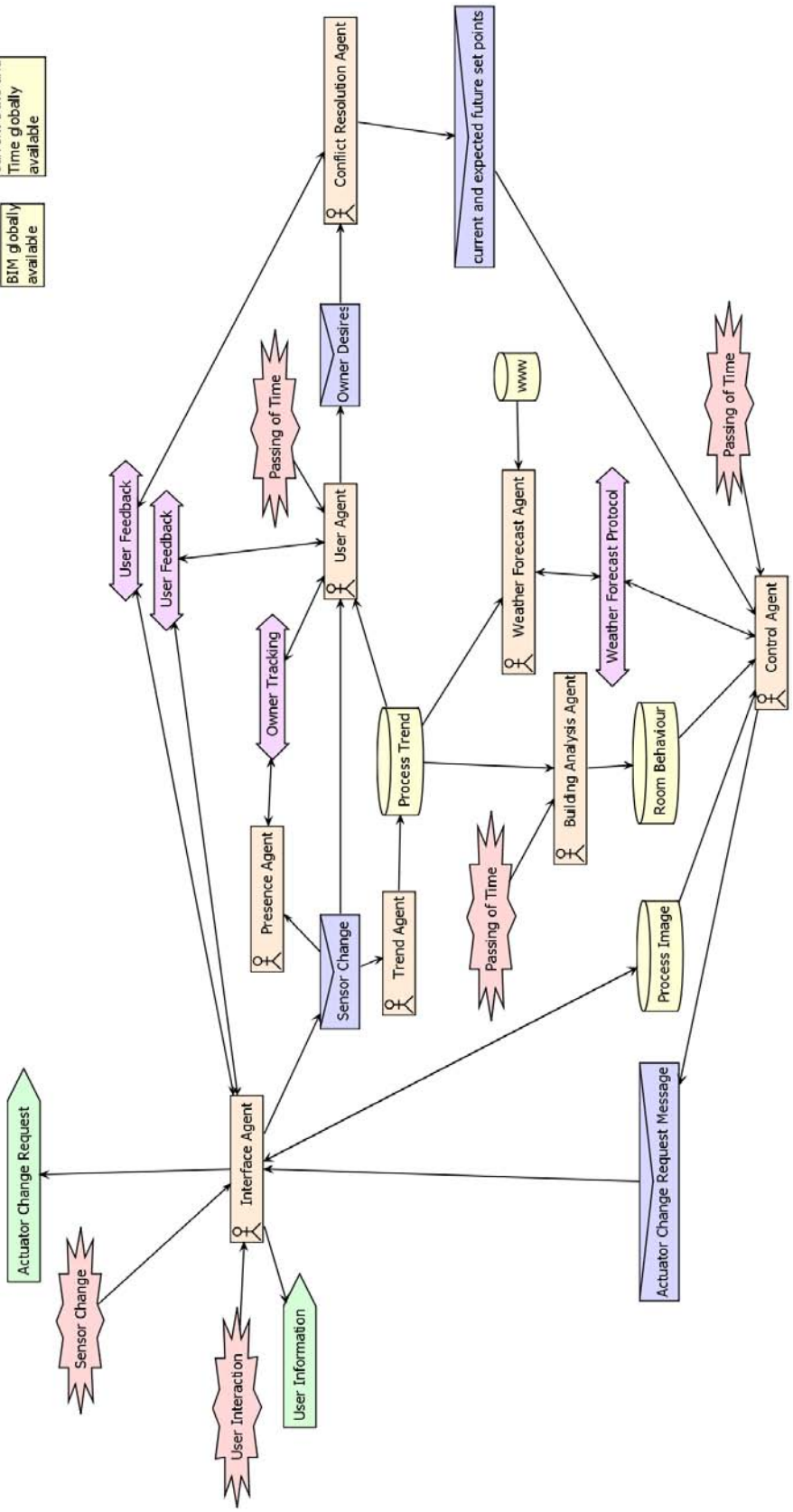


Figure 6: System overview

only. Therefore, our next steps will include to extend the simulation period to a full year. Also, we expect reducing the interaction interval from 15 minutes to one minute to provide further energy savings.

The Prometheus design methodology has been proven to be effective in assisting us to design our agent based system. We intend to embed the ANN controller as Controller Agent and write a wrapper for the environment simulation to be integrated as Building Analysis Agent. The implementation of further controller strategies is planned. Support vector machines and neuro-fuzzy controllers look promising and will be evaluated against the ANN controller. When finally put into real life, our agent system could be enhanced with further functionalities like lighting, multimedia and information services, elderly care, alarm services or some other future service like management and storage of energy from alternative power sources. Here, we have to meet the challenge of defining an appropriate building information ontology.

References

[1] US Department of Energy: http://apps1.eere.energy.gov/buildings/energyplus

[2] DesignBuilder Software: http://www.designbuildersoftware.com

[3] ATplus: http://www.modelica.org/libraries/ATplus

[4] R. M. Merz: *Objektorientierte Modellierung thermischen Gebäudeverhaltens*, Dissertation Universität Kaiserslautern, 2002. pp. 329-365, 1995.

[5] Dynasim AB: http://www.dynasim.se

[6] Thermal Energy System Specialists: http://www.trnsys.com

[7] B. Hayes-Roth: *An Architecture for Adaptive Intelligent Systems: Artificial Intelligence*. In Special Issue on Agents and Interactivity, 72,

[8] H. S. Nwana, D. T. Ndumu: *An Introduction to Agent Technology*. BT Technology Journal, 14(4), pp. 55–67, 1996.

[9] L. Padgham, M. Winikoff: *Prometheus: A Pragmatic Methodology for Engineering Intelligent Agents*. Proceedings of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies, pp. 97–108, 2002.

[10] Prometheus Design Tool: http://www.cs.rmit.edu.au/agents/pdt

[11] Jadex BDI Agent System: vsis-www.informatik.uni-hamburg.de/projects/jadex

[12] Java Agent DEvelopment Framework: http://jade.cselt.it

[13] Foundation for Intelligent Physical Agents: www.fipa.org

[14] W3C: http://www.w3.org/RDF

[15] W3C: http://www.w3.org/2004/OWL