An existing complete House Control System based on the REFLEX Operating System: Implementation and Experiences over a Period of 4 Years

Karsten Walther, Reinhardt Karnapke, Joerg Nolte
Brandenburg University of Technology
Cottbus, Germany
{kwalther, karnapke, jon}@informatik.tu-cottbus.de

Abstract

Today, even small residential buildings have a number of complex electrical devices that advocate the usage of automated control systems. But currently available systems are either hard to handle or expensive. This paper describes an automated house control system that has been in use for the last 4 years. It has been built using only freely available, inexpensive hardware and the open source operating system REFLEX.

1 Introduction

Factory and building automation become more and more important. Even for small buildings the number of devices that have to be controlled and the complexity of their interaction increases. As the number of available devices that can be used increases, not only the number of different control applications increases, but also different possibilities to implement these arise.

In this paper we present an existing house control application that has been in use for the last 4 years. In implementing this system, only standard, of the shelf hardware has been used. This reduces the total cost significantly, compared to existing complete solutions that can be bought. The control logic has been implemented using our own operating system REFLEX [5, 8]. This way, our control application is not only cheaper, but anyone interested is also able to customize the system without too much effort, because REFLEX is open source software.

This paper is structured as follows: Section 2 briefly describes the operating system we used, while section 3 shows the controlled elements and the hardware used to control them. Section 4 reports on some of the experiences we made and the lessons we learned. We finish with conclusion and future work in section 5.

2 The Reflex operating system

REFLEX (**R**eal-time **E**vent **FL**ow **EX**ecutive) is an object oriented operating system for deeply embedded control systems and sensor nodes. Applications are programmed according to the so called event flow model. In that model the schedulable entities are activities, that are triggered when events are posted to associated event buffers. Triggered activities are then invoked by the scheduler according to the chosen scheduling strategy.

Events can be associated with data but also dataless signals are allowed. They are raised either by interrupt handlers or other activities. The initial source of any activity in the system is always an interrupt. Figure 1 shows an example of an event flow graph.

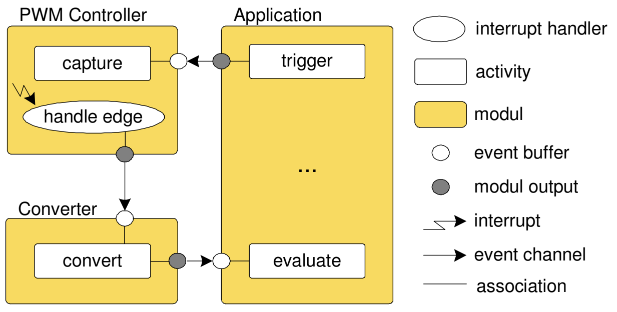

Figure 1. Event Flow Example for PWM Temperature Sensing

In the example the `trigger` activity of the application component signals the PWM-Controller (Pulse Width Modulation) to capture a value. The `capture` activity starts capturing. On signal edges the PWM module generates an interrupt. When the duty cycle of the signal is determined, this value is propagated as a raw value to a conversion component. This component transforms the raw value to a temperature, which is propagated to the application component. Finally, the `evaluate` activity evaluates the temperature and reacts accordingly.

It can be seen in the example, that components are containers for activities and interrupt handlers which provide a standardized interface. Typically components represent device drivers or processing stages. The communication between the components is always asynchronous, while inside the components the activities and handlers can share state information. This concept is similar to TinyGALS (Globally Asynchronous Locally Synchronous) [2].

The event buffers at the inputs of the components are needed for asynchronous communication. REFLEX already supplies standard buffers such as event counters (for signals), queues, fifos or simple value buffers. The access to the event buffers is always atomic for readers and writers. Most applications are therefore implicitly synchronized [5]. Furthermore all buffers count the posted events, the related activities are scheduled exactly once for each post. Thus the application code can rely on valid data at execution time.

The event flow model makes no assumption about the applied scheduling scheme. Details of the scheduling schemes are hidden in the system part of an `Activity` object. Therefore the choice of a given scheme is transparent for the inner implementation of the activities. The reusability of software components is therefore significantly eased and applications can often be composed of prefabricated bricks.

All activities have run-to-completion semantics like in other event driven systems. However, since activities are objects (instances of C++ classes) rather than functions, they can easily preserve important state information across multiple activations without the need for a private stack. So it is possible to implement a wide range of scheduling algorithms for the single stacked REFLEX system. The overall scheduling framework was already presented in [8]. So far we implemented FCFS- (First Come First Served), FP- (Fixed Priority), EDF- (Earliest Deadline First) and TT-scheduling (Time Triggered). The FP- and EDF-scheduler exist as preemptive and non-preemptive versions. Note that the single stack approach significantly lowers the memory requirements of an application, especially if a preemptive scheduling scheme is used.

3 The House Control System

There are two main parts in the house control system, the heating control and the yard light/alarm system components. The heating system consists of an oil burner, coal-burning kiln, sun tracking solar panel, room heaters, two warm water reservoirs, various pumps and temperature sensors. The yard light/alarm system consists of Reed-switches at the doors and gates, lamps, a light sensor and a siren.

Both subsystems work with the time of the day, which a DCF77-Clock component gets from an external battery buffered radio-clock module. There is also a standard output and the obligatory watchdog driver. All in all there are nine machine driver components, five of them are interrupt driven. Furthermore, the system has 13 machine independent components. Overall the system monitors 24 sensors, controls 16 actuators and communicates with 6 external devices.

3.1 The Heating Control

The hot water needed for showers, dish washing etc. is stored in 2 water reservoirs installed in the house, which can hold a maximum of 860 liters total. It is generated by 3 different sources: A solar panel, a wood burning kiln and an oil burning kiln.

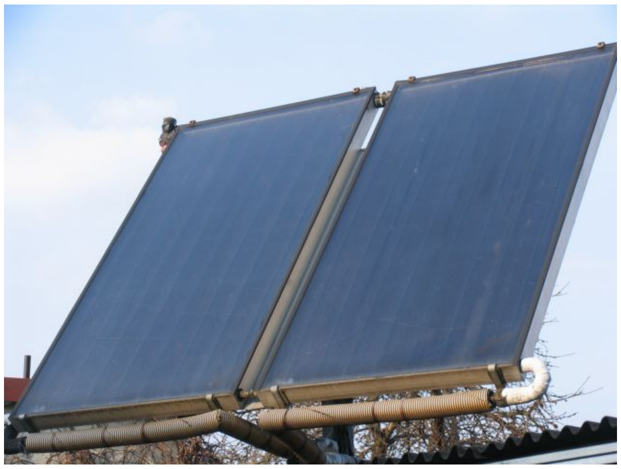

Figure 2. The moveable solarpanel

The solar panel of type Phönix Solar [6] used as primary heating system has a size of 3 square meters (figure 2). As the sun moves along the horizon, the amount of energy generated by a fixed solar panel would vary much, depending on the angle of impact. Therefore, the solar panel is attached to its supporting pole in a way that keeps it moveable. The position of the panel is changed each hour, according to the relative position of the sun. The best position for each hour has been determined a priory, because there is no substantial change in the position of the sun at a certain time of day between one day and the other.

Moving the solar panel must always be possible to produce the best heating results. The motor that moves the panel has its own independent power supply of 24 watts at 12 volts. End switches have been placed at the end of the range of the solar panel, to prevent it from turning to far and cutting its own cable. These are necessary because of the inherent inaccuracy of the motor. This inaccuracy could be reduced at additional cost, but using the touch sensors to correct the position is much easier and cheaper.

The hot water needed by the heating system is produced by two kilns, one burning wood (VEB Niederkirchner Berlin, figure 3, left side) and one burning oil (Hansa, figure 3, right side). In contrast to common systems where the heating constantly supplies hot water even if none is used, a usage oriented control system similar to the bajorath [1] principle has been implemented. Using a heat sensor, the temperature of the water flowing back is measured to determine the amount of energy absorbed inside the house. The usage oriented control system reduces the number of starts for the oil burner. Reducing the number of starts saves much energy, because in the start phase the efficiency of the burner is at its lowest.

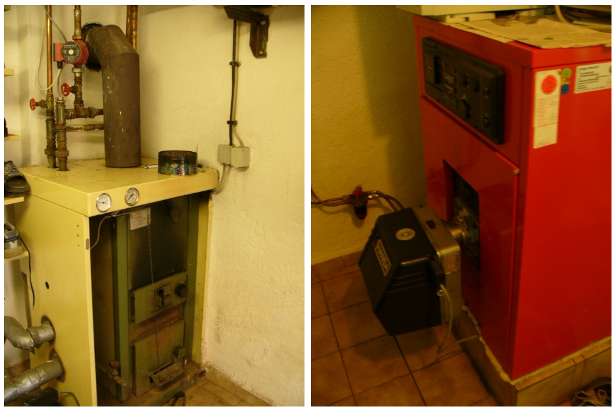

Figure 3. The two Kilns, burning wood (left) and oil (right)

While both of the kilns supply heating, their features are quite different and therefore their usage varies. The oil burner is completely controlled by the house control system. It can deliver hot water which is them pumped through the heating system almost immediately. But the cost of oil is high.

On the other hand, the wood burning kiln can only be monitored, it has to be fired up manually. But the costs for the wood are almost nonexistent, as the owners of the house have access to a forest in which they can cut as much wood as they need.

Therefore, the two kilns have been connected in a row, first the wood burning one, then the oil burner. A heat sensor measures the temperature of the water when it leaves the wood burning kiln and enters the oil burner. If it is already hot enough, there is no need to burn additional oil.

Figure 4. The frost protected doghouse

One of the more unconventional elements of this house control is the doghouse. As the temperatures lower significantly in the winter, the dog should still not freeze. Therefore, the doghouse is protected against frost by an electric heater (figure 4) combined with a temperature sensor.

3.2 The Yard Light/Alarm System

The layout of the courtyard is depicted in figure 5. As already described, the two kilns are connected in a row, which can be seen in the upper middle of the figure. Previously, the building on top has been a stable, but nowadays it houses the kilns, storage for wood and oil, garage, and laundry. The building on the right is an aviary where quails and parakeets are raised. The residential house is located on the left side, and the lower part of the figure shows another shed where bikes, lawn mower etc. are stored.

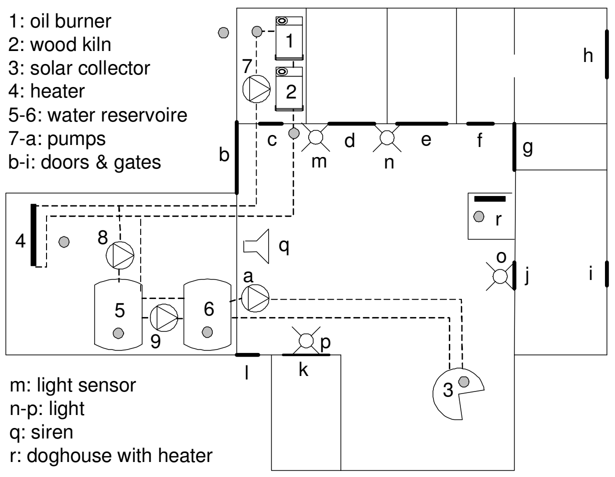

Figure 5. The layout of the house and courtyard

The middle of the figure shows the places where the lights (n-p) and the light sensor are affixed. When it is dark outside and one of the doors is opened, the light is turned on. Two of the lamps remain on for 20 seconds even after all doors are closed, to enable walking from one end of the courtyard to the other with light. The other (p) is turned off immediately.

The doors are also monitored for the alarm system. Most of the time, they are only being opened at certain times, in this case between 06.00 and 22.00 o' clock. This alarm system serves a dual purpose. First of all, it announces the unauthorized opening of doors after a certain time. Moreover, it also serves as a remainder for the inhabitants, to close all doors. If one of the doors, e.g. the one on the backside of the shed, is left open, this is not visible from the residential house. If it is still open at 22.00 o' clock, the sirens reminds the owners to close it.

The whole system is event-based, meaning that no polling of devices is necessary. Rather, there is a finite state machine behind the implementation, which only reacts on events.

3.3 The Main Control Unit and other Hardware

The main control unit consists of 6 elements. The power for sensors and control relays is provided by a power supply from Egston [3], which delivers 12 watts with 5 volts or 12 volts.

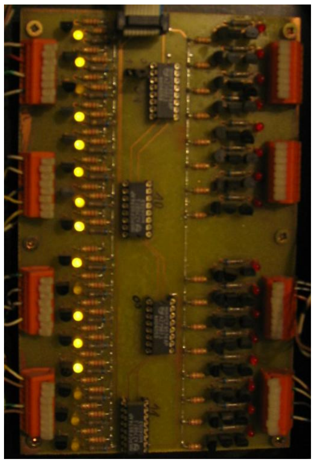

Figure 6. A Philips PC8574 I/O Expander

Digital inputs and outputs are connected via 4 Philips PC8574 I/O Expander [7] (figure 6), 2 of them for outputs and 2 for inputs. The outputs are used to control the light on the yard, as well as the electrical heating in the doghouse and the motor which adjusts the position of the solar panel. The inputs receive data from the door contacts, the light sensor and the end switches of the solar panel.

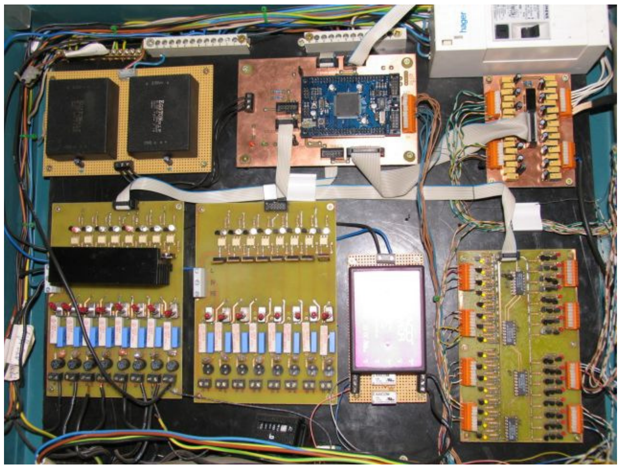

Figure 7. The main control unit

The circuit board in use is an Elektronikladen CardD64 Freescale HC(S)12 with 4KB Ram and 64KBRom, running with 8MHz. It can be seen on figure 7 in the upper center.

There is a number of additional hardware that is needed to keep the system running. To keep the oil burner and all of the pumps going, 16 Load switches based on Triac 220V/4A are used (figure 8). These have the additional advantage that they can be manually controlled. If an error occurrs, they can easily be deactivated.

The temperature sensors that are used to determine the heat levels at the kilns and in the house are of type

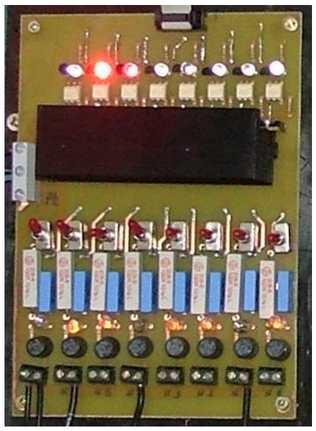

Figure 8. The Triac 220V/4A Loadswitches

SMT160. They generate a PWM signal with a frequency that fluctuates between 1 and 8 KHz. They are connected to the system using a Maxim 16x1Multiplexer.

As both the heating system as well as the light/alarm system are sensitive to the time of the day, a Conrad DCF77 radio clock is used to deliver the current time. It is battery buffered and connected to the microcontroller which emulates a parallel port. This emulation necessitates the usage of a priority based scheduling algorithm, as the parallel port needs a guaranteed response time of 0.5 milliseconds. The internal clocks are synchronized with the DCF77 every hour or after each reset.

The light/alarm system uses a siren and a light sensor from Conrad electronics. There are also various pumps at the kilns, in the heating circuit, for warm water, for oil and to circulate between the two water reservoirs.

The whole control unit is connected to a PC for monitoring and reprogramming using a RS232 connection.

3.4 Software Details

The whole software of the system is shown in figure 9. The dark grey components are hardware dependent drivers, while the light grey components are hardware independent. One of the most complex components is the DCF77 clock, because it has to emulate a parallel port. This single component contributes 1.7 kbyte to the code size. Another, even larger though simpler component that can be seen is the one that converts pwm signals to temperature values. While it is not complicated in itself, it needs floating point arithmetics, which contribute another 4 kbytes of library code to the system.

The whole system consists of approximately 800 lines of code. The total memory consumption is about 23 kbyte for code and 2 kbyte for data on the used HC(S)12 microcontroller. Realizing such a complex application with so few lines of code and only so little memory consumption was only possible because we used the REFLEX operating system as basis. The event flow principle removes the need for explicit synchronization, which eases programming a lot. Also, the event driven nature of REFLEX removed the necessity for polling algorithms.

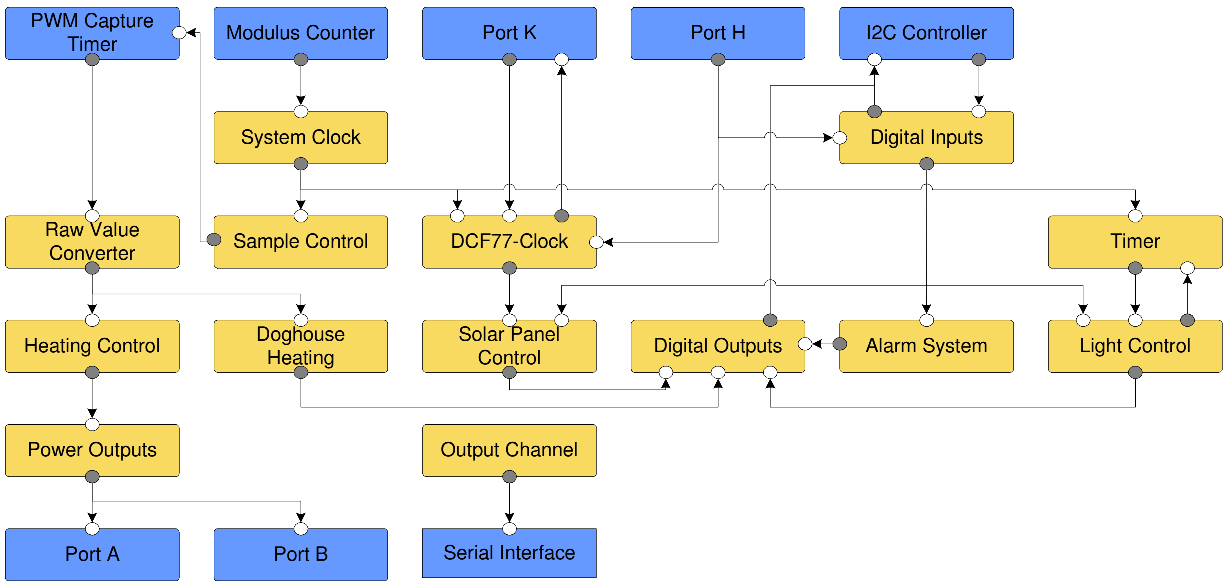

Figure 9. The house control as event flow diagram for REFLEX

4 Experiences

At the beginning of the 4 years, there was still an analogous board (figure 10) included in the control unit. It was connected to heat sensitive resistors, which were used to measure the temperature. But this setup proved to be inefficient soon, because the measured values were too inaccurate. Nowadays, the resistors have been replaced with digital temperature sensors.

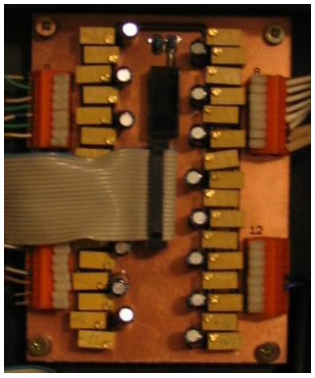

Figure 10. The analogous board used with the resistors

When the light control was first implemented, all three lamps stayed on for 20 seconds after the last door had been closed. This did not cost much more power, but had an interesting psychological effect. The owners of the house were not always convinced that the light control worked, they waited at the door until all three lights had turned off. For this reason, one of the lights was chosen as a kind of indicator. Now only two of the lights stay on, and the third is turned off immediately, reassuring the owners that the system is still working properly.

This problem existed for many of the aspects of a house control. Wherever direct interaction between humans and the system takes place, some sort of feedback must be supplied.

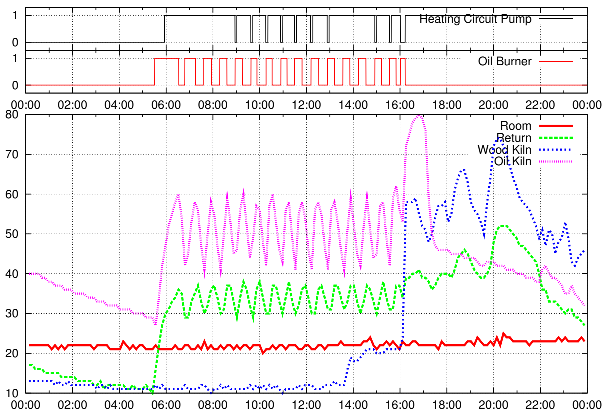

Figure 11. A Plot of the Temperature Levels in the Heating System for one Day in November 2007

Figure 11 shows an example of the heating system for a day in November last year. Until 6 in the morning no heating is needed. The upper curve denotes the temperature of the water after passing the oil burner. At night, it constantly cools down, until the system starts the burner at 6 am. Then the temperature of the water rises. Consequently, the temperature of the water flowing back from the house (curve below that of the oil burner) increases, too. As the heaters inside the house start to drain the warmth from the water to keep the house warm, the temperature of the back flow decreases again, leading to more oil burned.

At 13.30 one of the inhabitants went to the wood burning kiln and tried to fire it up. As can be seen on the figure, something went wrong and the temperature at the wood burning kiln increased only from 10 to 20 degree. Normally, it takes the wood burning kiln about 2-3 hours to burn a complete load. At around 16.00 one of the inhabitants went to refill it, and realized that it had not burnt properly. This time, firing it up worked and the temperature at the wood burner increased to around 65 degrees Celsius. As the oil burner was firing away at the same time, this lead to a peak in the water temperature of nearly 80 degree Celsius. Anytime the wood burning kiln reaches a temperature of 40 degree or higher, the oil burner is turned off, which was also the case here. But as it was already hot, it took some time to cool down.

Over the whole day, the temperature in the house varies only slightly. The set-point temperature for the house was 21 degrees during the day and 18 degrees at night. As can be seen the temperature is constantly above this desired value, if only a little. One more thing becomes evident when looking at the chart. Firing up the wood burner again at around 20.00 would not have been necessary, as even without it the set-point would have been kept.

At the beginning of the 4 years, the software needed to be changed nearly on a weekly basis, until all parameters were tuned. After that, the system was stopped from time to time to add features, e.g. code that eased monitoring and debugging. The monitoring software was especially important, because it was used to prove to the customers that the system was not malfunctioning as they believed. Rather, it was working just as they had specified, but the specification had been false. Moreover, effects like the useless firing up of the wood burning kiln could only be made visible with these monitoring tools.

Another feature was to differentiate between individual doors for the alarm system, as the garage sometimes needed to be opened before 6 am. Therefore, its timing had to be modified.

The system ran stable for longer periods of time, and was often only changed because we introduced a new version of our operating system REFLEX (see section 2).

The longest uptime of the system was 7 months. This uptime would still have increased, as there were no problems with the system. It did not crash. Rather, nature took its course. In summer 2007, just as these 7 month of uptime were reached, the system was destroyed by lightning that killed all electrical devices in the house.

5 Conclusion and Future Work

We have presented an existing house control application consisting of 24 sensors and 16 actuators which communicates with 6 external devices and has been in use for 4 years. This application has been realized using standard of the shelf hardware and a HC(S)12 microprocessor as main control unit, which makes using such a solution much less expensive than a conventional, PC-based approach. A PC or Laptop can be attached for monitoring and reprogramming, but there is no need for a permanent connection.

The house control has been realized using the open source operating system REFLEX, which made using the HC(S)12 microcontroller possible in the first place because of its small memory footprint. It also eased the programming a lot, because of its event driven nature and the inherent synchronization from the event flow principle.

Even though object oriented programming has eased development of the house control application significantly, for complex systems a high level modeling approach is desirable. Therefore, a commercial SDL tool has been adapted to a REFLEX runtime platform [4].

In the future, the connection between microcontroller and PC will be migrated from RS232 to Ethernet. Also, it could be possible to receive better results from the solar panel if it was affixed not only movable on one axis, but on two. The reason for this is that in winter the sun does not rise as high as in summer, therefore the angle of impact is different. It could be useful to change the vertical alignment during the year.

The most precious part of the system at the moment is the DCF77 clock, because it was the only available battery buffered radio clock that could be read via parallel port. It is also single point of failure, because both subsystems rely on the clock. In the future, it would be good to have a solution that does not depend on this special device.

A feature request from the users is to somehow integrate the weather forecast into the system. At the moment, when the temperature of the water in the storage falls below a certain threshold, the oil burner is started to keep the water warm. This may happen in the morning of a sunny day, as the system does not know anything about the weather. If it did, the oil burner could stay turned off, because the water would be heated by the solar panel within the next few hours, thus saving more of the expensive oil.

References

[1] R. Bajorath. http://www.bajorath.de.

[2] E. Cheong, J. Liebman, J. Liu, and F. Zhao. Tinygals: A programming model for event-driven embedded systems. In *SAC*, pages 698–704, 2003.

[3] Egston. http://www.egston.com/en/index.php.

[4] M. Jersak, K. Richter, R. Henia, R. Ernst, and F. Slomka. Transformation of sdl specifications for system-level timing analysis. In *CODES '02: Proceedings of the tenth international symposium on Hardware/software codesign*, pages 121–126, New York, NY, USA, 2002. ACM Press.

[5] K.Walther and J.Nolte. Event-flow and synchronization in single threaded systems. In *First GI/ITG Workshop on Non-Functional Properties of Embedded Systems (NFPES)*, 2006.

[6] S. Panel. http://www.phoenixsolar.com.

[7] Philips. http://www.nxp.com/acrobat/datasheets/pcf8574_4.pdf.

[8] K. Walther and J. Nolte. A flexible scheduling framework for deeply embedded systems. In *In Proc. of 4th IEEE International Symposium on Embedded Computing*, 2007.