# A Clock Synchronization Skeleton Based on RTAI

**Yang Huang, Peter Visser, and Jan Broenink**
University of Twente
Enschede, the Netherlands
Y.Huang@alumnus.utwente.nl
P.M.Visser@ewi.utwente.nl
J.F.Broenink@ewi.utwente.nl

**Abstract**

This paper presents a clock synchronization skeleton based on RTAI (Real Time Application Interface). The skeleton is a thin layer that provides unified but extendible interfaces to the underlying operating system, the synchronization algorithms and the upper level applications in need of clock synchronization. The skeleton provides synchronization support to a system, whereby the achieved accuracy is the best obtainable given this software structure. By connecting an algorithm and a communication module with the skeleton, a system becomes capable to run with synchronization support. To demonstrate and validate the design, the skeleton has been tested successfully with two different synchronization algorithms based on the CAN bus. Other algorithms and communication technologies can also work with the skeleton, as long as they provide the necessary functionalities for clock synchronization.

## 1 Introduction

A distributed system is a collection of independent computers that appears to its users as a single coherent system [1]. However, due to the difference between the independent computers in the software application, the hardware platforms, and working environments, local clocks in these computers will deviate from each other. To achieve monolithic performance, a distributed system needs clock synchronization support, which is a key issue when constructing such a system.

A lot of research has been performed to provide an accurate and efficient software algorithm for the clock synchronization between different nodes in a distributed system. These algorithms include the classical Cristian's Algorithm [2], the Berkeley Algorithm [3], the optimal clock synchronization algorithm [4] and the algorithm providing self-stabilizing clock synchronization [5] etc. These algorithms realize the clock synchronization of a system in different manners and have been successfully applied in many applications.

However, in order to realize a synchronization algorithm in a distributed system, one has to implement not merely the algorithm, but also the corresponding mechanism in the system, including the synchronization message transmission, synchronized clock supply etc. The mechanisms in different applications are similar to each other, independent of the communication technologies and the algorithms. To avoid this repeated work, a thin layer including these mechanisms to achieve clock synchronization is needed. With the layer, a synchronization algorithm can be applied directly in a system by only describing the algorithm itself. The essential and common functionalities are provided by the layer. On the other hand, in case the communication layer is changed, the algorithms are unaffected. The corresponding interfaces between the algorithms and the platform need to be modified. In this way a modularized synchronization scheme is realized.

In this paper such a layer is presented, as a wrapper around the synchronization algorithms. It is called the *Synchronization Skeleton* and provides unified but extendible interfaces both between the user applications and a synchronization algorithm, and between the algorithm and the underlying communication layer. Therefore, a synchronization algorithm can be implemented independent of the outside world. Also, modification of either user application or communication mechanism will not effect the functioning of the algorithm. RTAI (Real-Time Application Interface) is used as the real-time oper-

ating system for the implementation of the *Synchronization Skeleton*.

In section 2, the synchronization skeleton is presented. Section 3 shows the experimental results, and conclusions are drawn in section 4.

# 2 Synchronization Skeleton

In this section, the conceptual description gives an overview of the *Synchronization Skeleton*, followed by a detailed introduction of the entire mechanism, including its structure, the clock flow in the mechanism and the analysis on the clock accuracy of the skeleton.

## 2.1 Conceptual Design

The component diagram of the skeleton is shown in Figure 1.

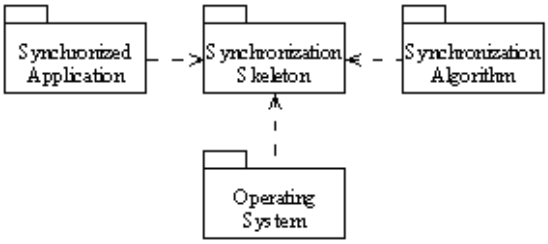


**FIGURE 1:** *Component Diagram of the Synchronization Skeleton*

From the top point of view, the *Synchronization Algorithm* describes the policy to realize the clock synchronization, while the *Synchronization Skeleton* provides the *Synchronized Applications* with the globally synchronized clock of the entire system. By reading the clock, applications in different nodes of the system will be kept synchronized with each other. The basic functionalities such as task scheduling, mutual exclusion, timing and communication between nodes are provided by the *Synchronization Skeleton* based on the *Operating System*. The *Synchronization Skeleton* is connected to the other three packages by a set of programming interfaces. Modification of *Synchronized Application*, *Synchronization Algorithm* or the underlying communication technology in the *Operating System* will not affect the other components. Only the corresponding interfaces in *Synchronization Skeleton* need to be changed.

## 2.2 Structure of the Skeleton

The structure of the *Synchronization Skeleton* is depicted in Figure 2. Except the *Synchronization Skeleton*, the blocks in the kernel space belong to the *Operating System* in Figure 1.

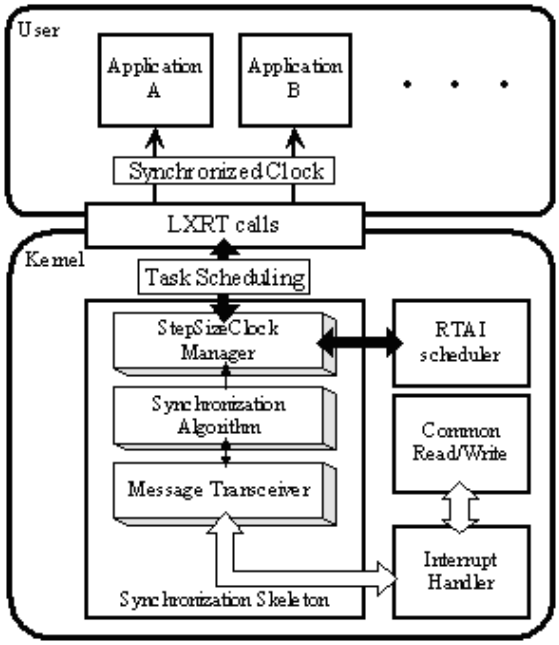


**FIGURE 2:** *The Structure of Synchronization Skeleton*

Located in the middle of the skeleton, the *Synchronization Algorithm* communicates with equivalents in other nodes via the *Message Transceiver*. The *Message Transceiver* is the interface of the skeleton to the communication layer. By connecting the transceiver with the communication layer, the corresponding communication functionalities in the operating system are exported to the skeleton. Because of this transceiver, the skeleton is able to work independently of the underlying communication technology. Clock synchronization in the skeleton requires deterministic message transmission. This feature, however, depends on the communication layer outside the skeleton. Thus for working with the skeleton, a real-time capable communication technology is needed.

The *Interrupt Handler*, which belongs to the communication layer of the operating system, acts as a message dispatcher. It decides whether a message is a synchronization message. For receiving / transmitting a synchronization message, the *Interrupt Handler* cooperates with the *Message Transceiver* directly. In this situation the *Common Read/Write* block of the operating system will not notice the message transmission. Otherwise the transmission is handled by the *Interrupt Handler* and the *Common Read/Write* blocks.

Within the *Synchronization Algorithm*, algorithm dependent calculation and signal processing is done. From the *Synchronization Algorithm*, a globally synchronized clock called *Step Size Clock* is pro-

vided to the *StepSizeClock Manager*, which is a tiny RTAI periodic task that resides in the core of the *Synchronization Skeleton*. The *StepSizeClock Manager* takes the clock and keeps itself synchronized with counterparts in the other nodes. In the following discussion, the notation $SSC(t)$ is adopted to denote the value of a *Step Size Clock* in the *Synchronization Skeleton* at real time $t$.

The *StepSizeClock Manager* helps to schedule all the user space applications according to their specific periods. In such a way, applications running above the *StepSizeClock Manager* are assured to be synchronized in the scope of the entire system. From the viewpoint of the *RTAI scheduler*, the *StepSizeClock Manager* is a proxy deciding at what time and which RTAI task is to be scheduled.

In user space, the application sees only a synchronized clock from the provided LXRT calls, which is called *rt_sync_task_wait_period*. Its name indicates the functionality: when an RTAI periodic task finishes its work within a cycle, invocation of the LXRT call suspends the task and the *StepSizeClock Manager* wakes it up when its period expires. In this way, a periodic task based on the synchronization scheme is realized.

## 2.3 The clock flow

The complete *Synchronization Skeleton* is designed to maintain a global clock in the system. The skeleton on each distributed node reads the local clock based on RTAI, calculates and provides the synchronized clock to the upper level applications. This can be illustrated by the figure below.

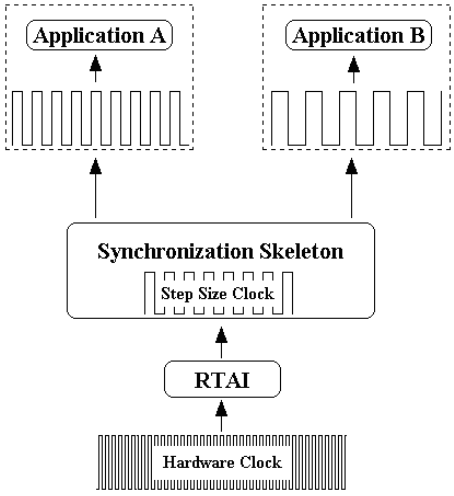


**FIGURE 3:** *The Clock Flow*

The hardware clock is read by RTAI, and the logical clock, *Step Size Clock*, is generated by the *Synchronization Skeleton* according to the configuration of the *StepSizeClock Manager*.

The *Step Size Clock* is the base clock in the skeleton and its period is configurable at initialization of the skeleton. Derived from the clock, different clocks can be generated and provided to the upper level applications. In this way, the skeleton can be regarded as a clock platform that offers clocks at different frequencies according to specific requirements in the user applications. Since the *Step Size Clock* is synchronized within the scope of the system, all the user applications are kept synchronized with each other in different nodes.

## 2.4 Accuracy of the skeleton

With the *Synchronization Skeleton*, all the nodes in a distributed system can be synchronized within a fixed clock deviation boundary. The accuracy of the clock synchronization is analyzed and determined.

Software synchronization algorithms are always discrete-update algorithms. Clock deviation between two successive resynchronizations is inevitable. Derived from the theory in [2], the maximum clock deviation, $d$, during period, $\Delta t$, between one node and the global clock can be expressed by

$$d = \rho \Delta t \tag{1}$$

where $\rho$ is the clock drift rate of the node. The maximum clock deviation between any two nodes in the system is then doubled as

$$d_{theory} = 2d = 2\rho \Delta t \tag{2}$$

where $d_{theory}$ is the theoretical clock deviation. In the real world the deviation is influenced by the environment factors. Three major factors are emphasized in the following discussion, namely the timer granularity, the deviation of the clock drift rate and the transmission time deviation.

1 Timer granularity

Each instance of *Synchronization Skeleton* maintains its own *Step Size Clock*. Since the *Step Size Clock* is a logical clock that increases discretely, its granularity has great impact on the synchronization accuracy. This effect can be presented in the following equation, taken from [6].

$$\frac{t - t'}{1 + \rho_i} - G \leq C_i(t) - C_i(t') \leq \frac{t - t'}{1 - \rho_i} + G \tag{3}$$

where $t$ and $t'$ are two different real time instances, $\rho_i$ is the clock drift rate of node $i$ in the system, $C_i(t)$ is the reading of the local physical clock of node $i$ at real time $t$, and $G$ is the clock deviation caused by the granularity.

Let the granularity of the *Step Size Clock* be $g$ , the time when the *Step Size Clock* ticks be $t_0$. For any time $t \in [t_0, t_0 + g)$, the equation $SSC(t) = SSC(t_0)$ always holds because of the resolution of the *Step Size Clock*. The minimum of $g$ here is system dependent and determines the synchronization accuracy of the skeleton.

When the effect of the clock granularity is considered, clock deviation between a local clock and the global clock, $d_g$, is determined by the larger value of $\rho\Delta t$ and the clock granularity $g$. That is

$$d_g = MAX\,[g, \rho\Delta t] \tag{4}$$

$g$ can be set at the initialization of the skeleton, and $\Delta t$ is also configurable with the knowledge of $\rho$. It is rational to set $g$ not larger than $\rho\Delta t$ and make it possible to detect and limit the clock deviation by adjusting the resynchronization interval $\Delta t$. On the other hand, the granularity of *Step Size Clock* is not supposed to be very small, otherwise the system will crash due to the heavy load. An optimal solution is achieved when

$$g = \rho\Delta t \tag{5}$$

which gives sufficient time resolution while more CPU power is left to other applications. Actually equation (4) becomes (1) when (5) holds, and the influence of clock granularity is gone. In this case, the $G$ in equation (3) can be regarded as zero and equation (2) holds.

2 Deviation of the clock drift rate

By measurement, the maximum clock drift rate in a system can be obtained. However, in reality the actual hardware clock drift rate is not always the maximum due to the changing environment. In this case extra clock deviation is introduced and should be taken into consideration when estimating the accuracy. Let $\rho_{max}$ and $\rho$ be the maximum drift rate of hardware clock obtained from measurement and the actual value, respectively. By equation (2) the minimum resynchronization interval is given as

$$\Delta t_{min} = \frac{d}{\rho_{max}} \tag{6}$$

With the knowledge of $\Delta t_{min}$ and $\rho$, the granularity of *Step Size Clock* is then as

$$g = \rho_{max}\Delta t_{min} \tag{7}$$

The assumption is made that the actual clock drift rate falls in the interval of $0.5\rho_{max} \leq \rho \leq \rho_{max}$ in the following discussion. The assumption is plausible as the drift rate will not change sharply given that the outside environment remains relatively stable. Derived from equations (1) and (7), $d = \rho\Delta t_{min} < g$.

As is shown in Figure 4, given that at time $t_0$, right after one resynchronization, a local clock is exactly the same with the global one. Before the next resynchronization the deviation of local clock is $d$, which is less than the period that can be detected. Due to the clock resolution, at $t_0 + \Delta t$ no resynchronization is done. This clock deviation is found at $t_0 + 2\Delta t$ , when it has grown to $2d$, which is larger than $g$. The resynchronization will be activated. However, the boundary of the actual clock deviation has already been violated.

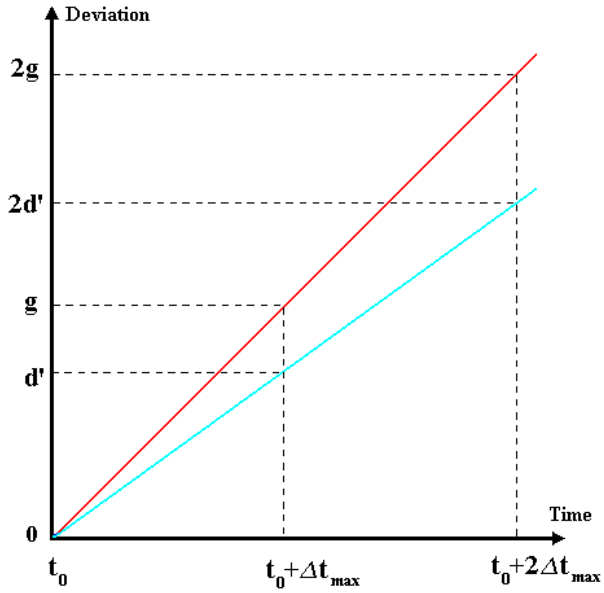


**FIGURE 4:** *Situation when MAX. clock deviation is violated*

Due to this effect, the clock deviation doubles again. As a conclusion, the total clock deviation between a local and global clock in the *Synchronization Skeleton* becomes

$$d' = 2d = 2\rho\Delta t \tag{8}$$

Then the clock deviation between any two nodes is

$$2d' = 4\rho\Delta t \tag{9}$$

3 Transmission time deviation

Another sort of deviation is introduced in the *Synchronization Skeleton*, due to deviation of synchronization message transmission. This part of deviation depends on specific communication technology and is expressed as $d_t$.

Taking all the aspects above into account, the clock deviation between any two nodes in the system is given as

$$D = 2d' + 2d_t = 4\rho\Delta t + 2d_t = 4g + 2d_t \tag{10}$$

for point-to-point synchronization message transmission, or

$$D = 2d' + d_t = 4\rho\Delta t + d_t = 4g + d_t \qquad (11)$$

for broadcasting the synchronization message.

With equation (10) and (11), the synchronization tightness of a system can be realized by setting $g$ and $\Delta t$ properly in the skeleton.

However, there are some exceptions when the actual clock deviation exceeds the maximum value. In the following situations, the clock deviation boundary cannot be guaranteed.

1 Large clock drift rate

The previous discussion is based on the assumption that by measurement the maximum clock drift rate in the system is obtained. In reality, this assumption may not hold. The actual clock drift rate would become higher than expected when the working environment changes considerably and is different with that in the measurement. For instance, when a distributed node starts up, the temperature of most electrical components in the node would be much lower than that in the normal working period. This may introduce a relatively higher clock drift rate. In case the real clock drift rate is larger than the value set in the skeleton, the synchronization tightness cannot be guaranteed by the skeleton. The corresponding strategy, however, can be defined in the synchronization algorithm to deal with this situation.

2 Heavy task scheduling

When multiple tasks are running in parallel on the same processor, and at a certain point of time more than one task need to be scheduled, the clock deviation boundary would be violated. This is due to the fact that on a single processor system, the RTAI scheduler together with the *StepSizeClock Manager* can only schedule one task at a time. In reality it is impossible to wake up more than one task in one node at the same time, and therefore, some tasks will unintentionally be delayed. In the worst case, when all the synchronized tasks are working at the same frequency and starting synchronously, which means all the tasks are to be waken up at the same time, there will be only one task be scheduled on time. The rest of them will be delayed. This delay time, therefore, should be taken into the total clock deviation, which is given as below:

$$D + (n-1)t_s \qquad (12)$$

where $t_s$ is the scheduling time for one task and $n$ is the maximum number of tasks, which need scheduling at the same time. However, it does not mean the system loses its clock synchronization. Only the clock deviation boundary is enlarged.

To ease the workload of the scheduler, one can either try not to have too many tasks running on the same node, or set a longer period for the *StepSize-Clock Manager*, which will decrease the synchronization tightness though. Another solution is to shift the execution time of the tasks in parallel a bit, such that the scheduling of these tasks will not happen at the same time.

# 3 Experimental Results

To test and demonstrate the performance of the *Synchronization Skeleton*, two synchronization algorithms have been implemented in the skeleton. The tests of the skeleton together with the algorithms have been made in several PC104 Embedded PCs, each of which has a VIA Eden ESP processor running at 600MHz. The operating system used was Linux 2.6 with RTAI 3.3. A real-time CAN driver was used for transmitting synchronization messages. Multiple user space processes were running on each PC and sending pulses at the defined frequencies via the parallel port, simulating the periodic behavior of distributed controllers. By observing and comparing the output pulses from the distributed nodes in a scope, the clock deviation in the system was obtained.

By measurement, the maximum clock drift rate in the system has been obtained as about 3 ppm (parts per million), and the deviation of CAN transmission in hardware was observed as 3 $\mu s$. Working in kernel space of the RTAI domain, software deviation in the skeleton can be ignored. To test the best synchronization tightness that can be achieved in the system, the granularity of *Step Size Clock* was set to 20 $\mu s$. Any value smaller will crash the system. By equation (10) and (11), the maximum clock deviation in the system is 83 $\mu s$ or 86 $\mu s$, depending on different algorithms.

The skeleton defines the best synchronization tightness of the built-in algorithms. Two synchronization algorithms have been implemented in the *Synchronization Skeleton* to demonstrate integration of algorithms in the skeleton.

## 3.1 The centralized algorithm

The first one is a centralized algorithm, which is a derivative of the Cristrian's Algorithm [2]. A synchronizer that holds the global clock of the system maintains the clock synchronization. By broadcasting the clock every resynchronization interval, the

synchronizer helps all the common nodes get synchronized in the system. Since the granularity has been set, the maximum clock deviation and the resynchronization interval is determined by equation (11).

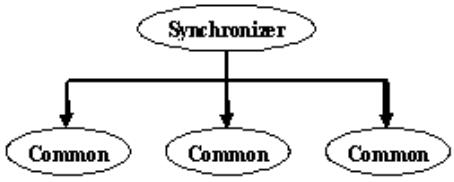


**FIGURE 5:** *The centralized algorithm*

### 3.2 The distributed algorithm

The other is a distributed algorithm, which is a simplified version of that described in [5].

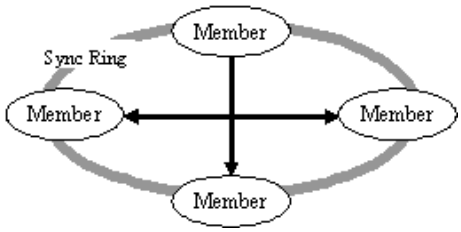


**FIGURE 6:** *The distributed algorithm*

In the distributed algorithm, each node in the system has the same opportunity to initiate a resynchronization. All nodes in the system form a *Sync Ring* in the system. Every node is a *Member* of the ring. With the knowledge of the node number and its position in the *Sync Ring*, a *Member* counts the synchronization messages it has received, by which the *Member* decides the time to broadcast the local *Step Size Clocks* as the global clock. This algorithm provides a more balanced system configuration since the implementation in all nodes is the same.

### 3.3 The results

The measurements have been performed using the setup with the two algorithms separately.

On each distributed computer there were 3 processes running in parallel and sending pulses to the parallel port at 1KHz, 500Hz and 250Hz respectively. Two groups of measurements have been done, with and without synchronization support. Also the two synchronization algorithms presented were tested. Based on the skeleton, both algorithms provide clock synchronization with the same accuracy. The measurement results are listed in Table 1.

| | Synchronization OFF | Synchronization ON |
|---|---|---|
| Clock Deviation in 10 min | 1.29ms | 72 $\mu s$ |

**TABLE 1:** *Clock Deviation Measurement*

Within 10 min the clock deviation between the distributed nodes was 1.29 ms without synchronization support. This is unacceptable for most distributed applications. In the other measurement, the clock deviation was kept within 72 $\mu s$, which is in the clock deviation boundary and will not grow with time. In the measurement it was also observed that the skeleton running on a single processor needs about 15 $\mu s$ to schedule one task. Therefore, except for the first task to be scheduled, the rest of the tasks running on the same node were delayed a bit.

The multi-node measurement setup is also used as a distributed control setup to control some mechanical systems for experimental purposes. Most controllers running in the setup are working at a frequency lower than 1KHz. The clock drift rate that is achieved with the skeleton is less than 10% of the controllers' working clock cycle. For such an application, the synchronization tightness is satisfactory.

## 4 Conclusions

This paper presented a thin layer called *Synchronization Skeleton*. The skeleton is inserted in a distributed system and provides unified and extendible interfaces to the clock synchronization algorithms, communication technologies and the user applications. In this way the realization of a new synchronization algorithm becomes much easier, and replacement of the underlying communication technology requires only modification of the interfaces. Besides, changing of either the algorithm or the communication technology will not affect the implementation of the upper level user applications.

The performance of the skeleton has been tested with 2 different synchronization algorithms in a multi-node distributed system based on a real time CAN driver. The test results showed that the *Synchronization Skeleton* together with the synchronization algorithms provides reliable clock synchronization support to the distributed control system.

Future work will be, but not limited to, the following:

Integration of more communication technologies and synchronization algorithms into the skeleton is attractive. This will bring the skeleton to a broader field of applications.

Integration of the skeleton into RTAI is appealing. The feature of clock synchronization is essential in distributed systems. Providing this feature directly from the operating system increases usability, and probably enhances performance.

Details about the implementation of the *Synchronization Skeleton* can be found in [7].

# References


[1] S.Tanenbaum, 2002, *Distributed Systems C Principles and Paradigms*, Prentice Hall, 0130888931.

[2] F. Cristian, 1989, *Probabilistic Clock Synchronization*, Distributed Computing, vol. 4, pp. 146-158.

[3] R. Gusella and S. Zatti, 1989, *The accuracy of the clock synchronization achieved by TEMPO in Berkeley UNIX 4.3BSD*, IEEE Transaction Software Engineering, vol. 15, pp. 847-853.

[4] T. K. Srikanth and T. Sam, 1987, *Optimal clock synchronization*, Journal of the ACM, vol. 34, pp. 626-645.

[5] A. Daliot, D. Dolev, and H. Parnas, 2003, *Self-Stabilizing Pulse Synchronization Inspired by Biological Pacemaker Networks*, Schools of Engineering and Computer Science,The Hebrew University of Jerusalem, Technical Report TR2003-1.

[6] F. Cristian, H. Aghili, and R. Strong, 1986, Clock Synchronization in the Presence of Omission and Performance Failures, and Processor Joins, 16th International Conference on Fault-Tolerant Computing, vol. 23, pp. 218-223.

[7] Y.Huang, 2006, Clock Synchronization using Real-Time CAN, Control Laboratory, University of Twente, Enschede MSc. Report, 015CE2005.