**Proceedings of the IASTED International Conference**
**Applied Modelling and Simulation**
**September 1-3, 1999, Cairns, Australia**

# A Unified Approach for Simulation and Experimental Evaluation of Fault-Tolerant Distributed Systems

ULRICH SCHMID, BETTINA WEISS, GÜNTHER GRIDLING, KLAUS SCHOSSMAIER
Technische Universität Wien, Department of Automation
Treitlstraße 1, A-1040 Vienna, Austria
Email: {s, bw, gg, kmschoss}@auto.tuwien.ac.at

## Abstract

This paper[1] surveys our framework for simulation and experimental evaluation of round-based clock synchronization algorithms in fault-tolerant distributed real-time systems. Developed in our project SynUTC, our toolkit is based upon a generic architecture that incorporates either real network controllers and clock devices or, alternatively, their simulated counterparts. Despite the higher design complexity, we obtained a considerable saving w.r.t. overall development time due to the fact that major parts of the toolkit, like data capture/analysis, are the same for both simulation and evaluation. We briefly outline the appropriate features and round off by a few sample results gathered from the simulation of two clock synchronization algorithms.

**Keywords:** discrete event simulation, clock synchronization, fault-tolerant distributed real-time systems, data visualization, performance evaluation.

## 1 Motivation

Dealing with distributed algorithms for fault-tolerant real-time systems usually requires more than one technique for validating a solution. In our research project SynUTC[2], for example, we employ analytical methods for deriving correctness proofs and worst-case bounds for precision and accuracy. Those results, however, do not answer questions about the average case behavior — apart from the observation that the worst-case is only attained in quite "exotic" execution runs. Nevertheless, there are interesting applications that primarily require an underlying clock synchronization service with good average accuracy.

Since in-depth analytical modeling of average case performance is difficult, either experimental evaluation or simulation must be used for assessment. Experimental evaluation has the advantage of inherently providing realistic results, but has a number of well-known drawbacks:

- The required hardware and software platform has to be at hand.
- Gathering representative data in real-time execution can take much time.
- There is only limited control of the behavior of system components.

Simulation helps to circumvent those problems, at the cost of setting up a suitable simulation tool. First of all, simulation studies can begin as soon as the algorithms are available. Second, the exploitation of simulation time instead of real-time considerably shortens the evaluation process. This is particularly true for round-based algorithms, where short phases of activity are separated by long (10 to 100 seconds) idling periods. Last but not least, simulation offers complete control over the behavior of the system with respect to parameters and fault occurences.

The SimUTC/EvalSys toolkit [1], [2] has been developed to investigate the average case performance of interval-based clock synchronization algorithms [3]. It provides a complete environment for both simulation and experimental evaluation of such round-based algorithms in networked distributed systems. In this paper, we give a brief overview of our toolkit, which is organized as follows: Section 2 introduces the target hardware, Section 3 explains the features of our toolkit, and Section 4 completes the description with two examples of practical usage.

## 2 Target Architecture

Our primary problem is to synchronize an ensemble of physical clocks that are spatially distributed. Each clock is part of an independent computing system called a *node*, which is connected to a data communication network. In order to achieve clock synchronization in such a setting, each node periodically broadcasts its own clock value to all other nodes in the course of a *full message exchange* (FME). Any node collects and processes the incoming messages from all the other nodes, and, near the end of a *synchronization round*, computes a correction value for its clock via

[1] Invited for AMS'99 special session on *Computing Systems Performance Monitoring, Evaluation and Visualization*.

[2] The SynUTC-project received support from the Austrian Science Foundation (FWF) grant P10244-ÖMA, the OeNB "Jubiläumsfonds-Projekt" 6454, the BMWV research contract Zl.601.577/2-IV/B/9/96, and the Austrian START programme Y41-MAT. For further reference see *http://www.auto.tuwien.ac.at/Projects/SynUTC*

a suitable *convergence function* (CF). Thereafter, at a specific point in time, the clock is adjusted, which launches the next synchronization round as shown in Figure 1. The quality of synchronization is measured by the *precision*, which is the maximum difference between the values of all pairs of clocks, and the *accuracy*, which denotes the difference between each clock value and real-time.

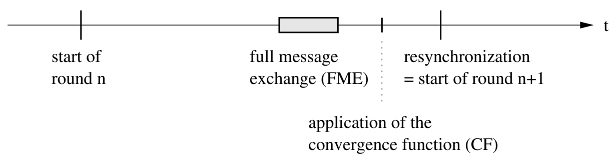


Figure 1: *Synchronization Round*

The SynUTC project targets clock synchronization with worst case precision/accuracy in the $\mu$s-range in LAN-based system architectures. This goal necessitates some hardware support, which is provided by our *Network Time Interface* (NTI) M-Module [4]. Apart from hosting the state- and rate-adjustable clock implemented in our *Universal Time Coordinated Clock Synchronization Unit* (UTCSU) [5], the NTI is primarily responsible for timestamping all messages that are sent from resp. received at a node. Since timestamping is actually done when a certain byte of the message is put on resp. obtained from the network, neither the varying execution time of our network device driver [6] nor the present network traffic affects timestamping accuracy. Thus, the atop running clock synchronization algorithm is relieved from all time-critical timestamping tasks, but is otherwise not affected by our hardware support.

The target system used for experimental evaluation is a VMEbus-based distributed system consisting of 16 nodes running under ISI's pSOS$^{+m}$ multiprocessor real-time kernel. Any node is equipped with an NTI, an Ethernet network controller and a CPU, which executes the SimUTC software. SimUTC consists of the clock synchronization algorithm under study and a monitoring environment that keeps track of its execution. System control and collection of monitoring data is performed by the evaluation system EvalSys running on a Linux workstation, which communicates with the nodes' SimUTC via TCP/IP stream sockets. Figure 2 portraits this target system architecture.

# 3 Evaluation System

When we started working on SimUTC/EvalSys, the target system hardware was not yet available. Therefore, we decided to build an evaluation framework that uses *pure simulation* initially, but allows switching to *hardware-based simulation*, that is, experimental evaluation, when the hardware (clock and network) is eventually at hand. It eventually allowed us to use the "best of both worlds" during the various stages of our project: Simulation was the

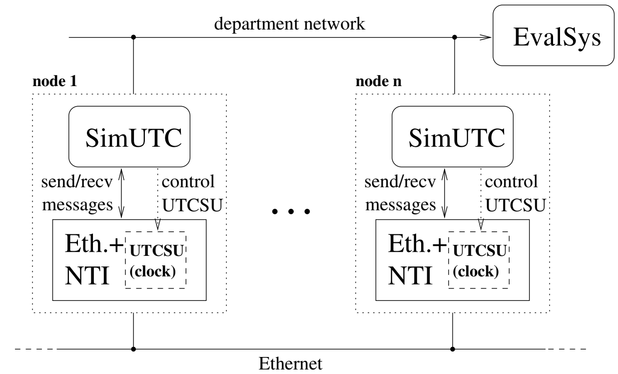


Figure 2: *Experimental Evaluation Architecture*

only means to get representative performance data during the initial stages, and proved advantageous later on due to its much faster execution and improved controllability as well. Experimental evaluation is now used to evaluate the actual performance of our prototype hardware/software implementation, which became available only recently.

SimUTC's flexibility primarily rests upon a suitable internal architecture and well-defined internal interfaces. It is written completely in C++ and can easily be extended by additional algorithms [7]. In pure simulation, it uses the discrete-event simulation package C++SIM [8] to model processes and interrupts, which are provided by the facilities of the pSOS$^{+m}$ kernel in hardware-based simulation. To enhance the toolkit's flexibility, we enforced a strict separation between the simulation system SimUTC and the X-windows based graphical user interface EvalSys. Simulation control is achieved by sending command messages over dedicated stream sockets. Similarily, event messages reported by SimUTC on these sockets are used to gather monitoring data from all nodes. Therefore, EvalSys allows the user to configure and command SimUTC and carry out data analysis regardless of the simulation type.

Upon the start of a session, EvalSys first builds the desired system configuration and initializes all parameters as specified. Once the environment is established, SimUTC starts sending event messages to notify EvalSys about all changes of the local system state. In order to make the states of all nodes comparable, SimUTC uses dedicated clocks to establish a global time (the "real-time") to periodically trigger a *snapshot*, i.e., the sampling of the clock state, concurrently on all nodes and sends this data to EvalSys. Since all messages are timestamped by the global time, it is easy for the evaluation system to construct the global system state and to extract characteristic quantities like precision and accuracy.

Figure 3 shows the user interface of EvalSys. The lower half of the application is consumed by the *event stream window*, which displays a user-definable selection of the SimUTC events reported by each node, sorted by the global

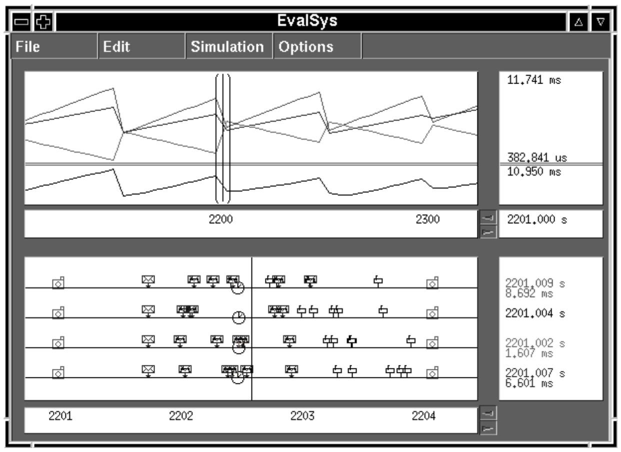


Figure 3: *Evaluation System User Interface*

time. SimUTC offers several event types, each of which contains specific information about the system state. Clicking on an event icon shows the data of this event. The user can display snapshot data at a certain time by setting a *position marker*, which is represented by the vertical line.

The *precision window* situated above the event stream shows the global (= maximum) precision in its lower part, and the accuracy of each node participating in the synchronization in its upper part. The accuracy graphs can be selectively hidden to avoid cluttering the display. Two curved vertical lines represent the part of the whole stream that is displayed in the event stream window. The position selected in the event stream is again marked with a straight vertical line.

The *state windows* to the right consist of a display of global data next to the precision window and a display of node-specific data next to the event stream window. Node-specific data consist of the clock value at the snapshot time and, optionally, its accuracy. The maximum and minimum accuracy encountered for the selected nodes are shown next to the accuracy graphs. Below, the worst case precision of the current simulation run is printed. The small window in the middle displays the simulation time of the last snapshot before the marker position.

At any time, the user can scroll over the event stream and zoom into it to get a more detailed view of interesting sections. Additional means of navigation are available, for example finding local or global maxima or minima in the precision graph. User-definable search conditions help to filter out or to find given system states.

A major requirement put on the development of EvalSys was maximum flexibility with respect to modifications of the simulation. Consequently, the layout of all message data structures used in the communication between EvalSys and SimUTC is loaded into EvalSys from text files at startup. System configuration itself can be automated via parametrized macros and saved for future sessions. In addition, EvalSys can print certain simulation data into files in *gnuplot* format [9]. The data is taken from the snapshot events and plotted over the global time. This feature allows further processing of the data with a simple text editor and to incorporate it into documents. In Section 4, we will provide a few examples:

- *Accuracy* plots the deviation of a node's clock value from the simulation time at which it was read.
- *Precision* gives the maximum difference of the simultaneous clock values of all pairs of nodes.
- *Adjustment* displays the difference between the old clock value and the new clock value enforced by the synchronization algorithm.
- *Accuracy interval* shows the accuracy interval of the clock at the time the clock was read.

Clearly, SimUTC/EvalSys provides full control over almost any system parameter. For example, the *algorithm* used in a particular evaluation can be chosen interactively from the list of implemented ones, and the timing characteristics of its synchronization round can be setup via three parameters: The *round period* specifies the duration of the full round, the *delay compensation* gives the time from the start of the FME until the application of the convergence function, and the *computation delay compensation* denotes the time from the application of the CF to resynchronization. Other parameters required for most clock synchronization algorithms are the *number of nodes* participating in the synchronization and the maximum *number of faulty nodes* that the algorithm can tolerate.

A pivotal component of SimUTC are the clocks, whose main parameters are the *clock state* and the *clock drift*. To facilitate accurate modeling of a typical clock's drift over time, we distinguish between the *initial clock drift*, which accounts for the inevitable (manufacturing-dependent) initial frequency offset of a hardware clock, the *random drift*, which models spontaneous drift "jumps" as caused by frequency noise, and the *deterministic drift*, which can be used for modeling dynamic drift changes due to deterministic effects like temperature changes.

In pure simulation, the network is under full simulation control as well. Its key parameters are the *transmission delay characteristics*, which consist of the deterministic and maximum probabilistic part of the transmission delay between each pair of nodes, along with an appropriate distribution function. Apart from the transmission delays, there are two additional delay parameters *broadcast latency* and *broadcast operation delay* that allow to model different network topologies. The former can be used to account for contention on shared channels like Ethernet, the latter applies to non-instantaneous broadcasting as in point-to-point networks.

Finally, SimUTC provides various fault injection capabilities that cause the simulation to occasionally transgress certain “normal operation” specifications. For example, the user may specify that messages of a certain node get lost upon transmission with some probability.

# 4 Examples

In this section, we present two simulation runs with our SimUTC tool. The first one utilizes the well-know *Fault-Tolerant Average* (FTA) clock synchronization algorithm [10], the second one the *Optimal Precision* (OP) algorithm documented in [7]. For both runs, the simulation model remains the same and is given by the following items:

- The ensemble consists of 5 non-faulty nodes.
- The respective initial clock drifts are $0$ ppm, $0.5$ ppm, $-0.5$ ppm, $-1$ ppm, and $2$ ppm. The dynamic drift changes are within $\pm 1.5$ ppm.
- The nodes are attached to a non-faulty broadcast network with a maximum latency of 1 ms.
- The transmission delay obeys a Normal distribution in the range of 50 $\mu$s to 80 $\mu$s with an expectation of 60 $\mu$s.
- The round period is set to 12 s and the delay compensations amount to 1.8 ms.
- The snapshot period is 3 s and the simulation terminates after 300 exchanged messages.

Figure 4 shows three plots gathered from the simulation run with the FTA function. The top one shows the accuracy of each clock, which clearly illustrates the periodic resynchronizations and drift phases in between. Moreover, we can verify the individual clock drifts, whereas the whole ensemble drifts away from the global time by approximately $0.4$ ppm.

The middle plot in Figure 4 shows the precision, which is mainly determined by the slowest node 4 and the fastest node 5. It has the expected sawtooth-like shape, where the endpoints are shifted according to the encountered transmission delay uncertainty. In this particular run, the precision wanders roughly between the 5 $\mu$s and 42 $\mu$s bounds.

Finally, the bottom plot in Figure 4 exhibits the adjustment of each clock at the resynchronization time. For instance, node 3 experiences an advancement of about 10 $\mu$s at every resynchronization instant. Note that all nodes measure the average round-trip delays of their messages before embarking with the synchronization, with nodes 1 and 5 starting early.

Figure 5 is made up of four plots obtained from the simulation run with the OP function, where the first three ones are analogous to Figure 4. The differences between the corresponding plots are only minor: The ensemble’s drift is about $0.6$ ppm, the precision is slightly better (8 $\mu$s and 39 $\mu$s bounds), and the adjustments are less regular. Of course, to actually compare the performance of synchronization algorithms, statistical data from sufficiently many runs must be collected.

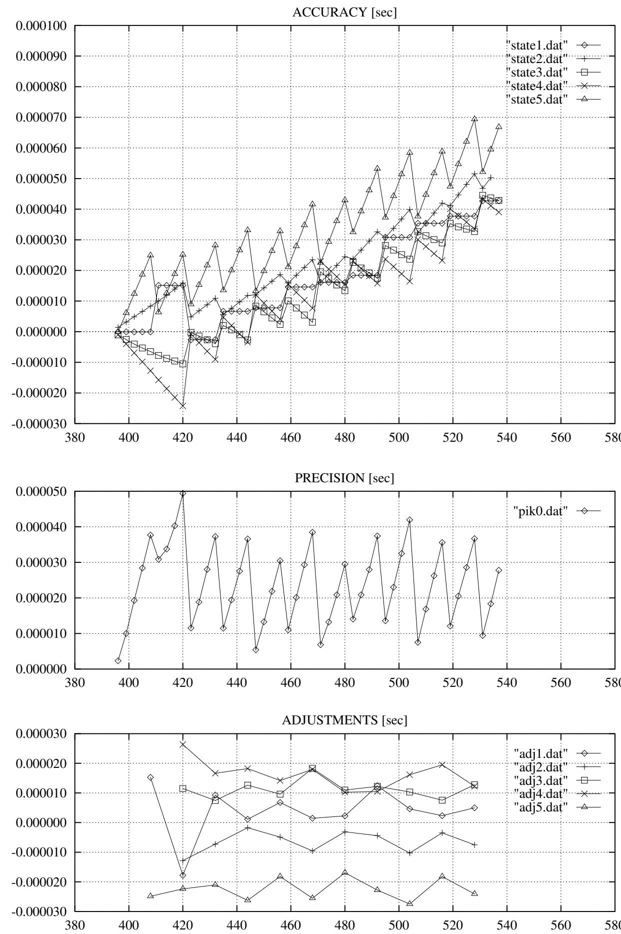


Figure 4: *Simulation Run using FTA*

The bottom plot in Figure 5 presents a novelty of our interval-based clock synchronization approach, see [3]. Its primary benefit lies in the fact that each node is provided with an on-line bound for its instantaneous accuracy. To that end, our UTCSU clock continuously maintains an *accuracy interval* that is said to be correct if it includes the (unknown) real-time. To keep up the correctness of accuracy intervals, they must be gradually enlarged (deteriorated) during the whole round because of the non-zero clock drift; in our example, we have chosen a deterioration of $\pm 3$ $\mu$s/s for each clock. During an FME, any node’s accuracy interval is distributed to all other nodes. From the resulting input, the convergence function OP can compute a new local accuracy interval that is put into effect at the beginning of the next round.

For clarity, our plot shows the accuracy intervals in conjunction with the corresponding clock’s accuracy and imposes a (small) fixed x-offset for each node to separate them at simultaneous snapshot times. Note that any correct interval must intersect the x-axis here. The average growth of the accuracy intervals reads $\pm 1.5$ $\mu$s/s after the first resynchronization has taken place.

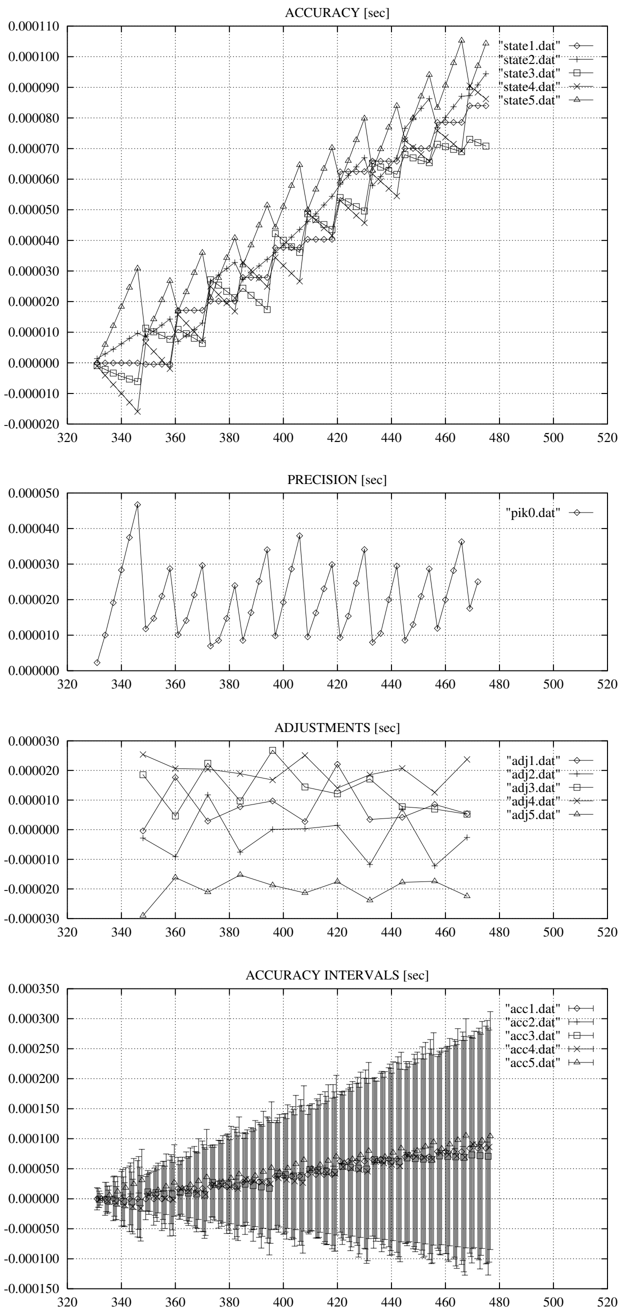


Figure 5: *Simulation Run using OP*

# Conclusions

We presented an overview of our SimUTC/EvalSys toolkit designed for the evaluation of round-based clock synchronization algorithms in networked distributed systems. Due to its modular and distributed architecture, SimUTC/EvalSys is a flexible and powerful tool for evaluating the performance of such algorithms both in a fully simulated environment and on a real target system.

Unlike the only related simulation framework [11] we are aware of, SimUTC provides elaborate simulation models for both network and clocks. In fact, when aiming at high accuracy clock synchronization in the 1 $\mu$s range, many system parameters ranging from clock granularities up to oscillator stability must be taken into account and, hence, appropriately modeled. Moreover, various fault-injection capabilities in conjunction with customized data analysis features facilitate in-depth simulation studies over long periods.

# References

[1] Bettina Weiss. Simulation environment for clock synchronization. Technical Report 183/1-88, Department of Automation, TU Vienna, February 1999.

[2] Bettina Weiss, Günther Gridling, Ulrich Schmid, and Klaus Schossmaier. The SimUTC fault-tolerant distributed systems simulation toolkit. Technical Report 183/1-94, Department of Automation, TU Vienna, April 1999.

[3] Ulrich Schmid and Klaus Schossmaier. Interval-based clock synchronization. *J. Real-Time Systems*, 12(2):173–228, March 1997.

[4] Ulrich Schmid, Johann Klasek, Thomas Mandl, Herbert Nachtnebel, Gerhard R. Cadek, and Nikolaus Kerö. A Network Time Interface M-Module for distributing GPS-time over LANs. *J. Real-Time Systems*, 17(2), 1999.

[5] Klaus Schossmaier, Ulrich Schmid, Martin Horauer, and Dietmar Loy. Specification and implementation of the Universal Time Coordinated Synchronization Unit (UTCSU). *J. Real-Time Systems*, 12(3):295–327, 1997.

[6] Gerda Richter, Michael Schmidt, and Ulrich Schmid. i82596 NTI device-driver software documentation. Technical Report 183/1-90, Department of Automation, TU Vienna, February 1999.

[7] Klaus Schossmaier and Johann Klasek. Implementing the optimal precision algorithm for clock state & rate synchronization. Technical Report 183/1-91, Department of Automation, TU Vienna, February 1999.

[8] Mark C. Little and D.L. McCue. Construction and use of a simulation package in C++. *C User's Journal*, 12(3), 1994. Postscript file available at http://cxxsim.ncl.ac.uk/homepage.html.

[9] Thomas Williams and Colin Kelley. GNUPLOT – An interactive plotting program. Program available e.g. at ftp://ftp.dartmouth.edu/pub/gnuplot/, 1993.

[10] Jennifer Lundelius-Welch and Nancy A. Lynch. A new fault-tolerant algorithm for clock synchronization. *Information and Computation*, 77(1):1–36, 1988.

[11] M. J. Pfluegl and D. M. Blough. Evaluation of a new algorithm for fault-tolerant clock synchronization. In *Proc. Pacific Rim Internation Symposium on Fault Tolerant Systems*, pages 38–43, Kawasaki, Japan, September 1991.

## Biographies

Ulrich Schmid received his diploma (1985) and Dr.techn. degree (1986) from the Technische Universität Wien (computer science and mathematics). Currently, he is associate professor at the Department of Automation at TU Wien. Dr. Schmid wrote numerous papers in the field of theoretical and technical computer science and received several best-paper awards and prices, like the Austrian START-price 1996. His current research interests focus on distributed algorithms, real-time systems and wireless computer communications. Ulrich Schmid is member of IEEE Computer Society and EATCS.

Bettina Weiss received her diploma in computer science from the Technische Universität Wien, Austria, in 1997. She has several years of experience as a teaching assistant at the Department of Automation at TU Wien, where she is now employed as a full-time research assistant working towards her PhD. Her primary areas of interest are real-time systems, distributed systems and wireless computer communications.

Günther Gridling earned his diploma in computer science from the Technische Universität Wien, Austria, in 1996. He has been working on several industrial and academic research and development projects, e.g. in the VLSI Design Group at TU Wien. Currently, DI Gridling is teaching assistant at the Department of Automation at TU Wien and about to start working on his PhD. His interests cover VLSI design, embedded systems programming and real-time systems.

Klaus Schossmaier earned a diploma (1991) and Ph.D. (1998) degree in Computer Science from the Technische Universität Wien, and an M.Sc. (1994) degree from University of Massachussetts at Amherst. His research interests lie in the areas of distributed computing and embedded systems, with emphasis on clock sychronization and real-time communication. He has been working in Project SynUTC and is about to join a data acquisition group at CERN. Dr. Schossmaier is a member of ACM and a recipient of the Fulbright Scholarship for 1992/93.