Poster Abstract: Enabling Real-Time in WSN Applications

© Marcel Baunach, Clemens Mühlberger, Christian Appold 2009
Department of Computer Engineering, University of Würzburg, Am Hubland, 97074 Würzburg, Germany
Email: {baunach, muehlberger, appold}@informatik.uni-wuerzburg.de

***Abstract*—Increasing complexity of today's WSN applications can quickly boost the real-time requirements of the underlying sensor nodes. Using preemptive operating systems with special scheduling, resource and timing mechanisms is one way to retain acceptable reactivity for single tasks, nodes and finally for the overall system.**

I. Why Real-Time matters

Current research on Wireless Sensor Networks (WSN) is still mainly focused on networking itself. Indeed, increasing performance and computational power of recent sensor nodes allows a much wider scope of useful application scenarios. Especially Sensor Actor Networks (SANET) are central for the upcoming Ubiquitous Computing but also establish new challenges and require a clear focus shift from pure communication towards real-time capability within such distributed systems. The inherently strong interaction with the surrounding environment as well as the fast cooperation of participating nodes and embedded components will commonly need a much higher level of system reactivity. Sensing, interpreting and propagating environmental conditions will not suffice any more, but active participation and proactivity gain in importance, too. Additionally, complex applications often apply a modular design in which even simple software compositions can rapidly result in serious runtime problems and significant performance loss of the overall system. Still, available operating systems for sensor nodes often address these problems just insufficiently. However, special support within multi-threaded preemptive systems might help to retain acceptable reactivity for individual nodes and the overall WSN. Initially, we will point out quite typical real-time aspects within a complex real-world scenario and describe the various emerging problems regarding task scheduling, resource and time management. Finally we will present our concepts and solutions within the fully preemptive real-time operating system *Smart*OS.

II. Requirements of Real-World Scenarios

In this section we will present the potential application complexity by presenting a real WSN based tracking and steering control system for indoor vehicles.

The central localization algorithm periodically estimates the vehicle's current position from distance measurements between a mobile node mounted on the vehicle itself and static nodes within the environment. Then, a fuzzy controller computes the subsequent deviation from the desired track and readjusts the vehicle movement. The core software system comprises seven concurrently running main tasks for distance measurement, position estimation, radio communication, fuzzy logic, motor control as well as the main application and a remote maintenance task for software updates. Nevertheless, even a single one of today's low performance sensor nodes can be sufficient if the various emerging problems from the application's compositional complexity are handled adequately. The prerequisites will be addressed next.

Since the vehicle perpetually moves around, at least localization and fuzzy control must be implemented as *periodic tasks* with a suitable frequency to avoid crashes at the current velocity. The distance measurement in particular, which is based on TDoA between radio and ultrasound, requires a constantly precise, high-resolution and system load independent *timestamping* of signal arrival times. This also implies the timely allocation and configuration of the sender and receiver hardware, which requires *resource sharing* among several tasks. The aggregation of distance information at the mobile node uses a TDMA radio protocol and thus requires node *synchronization* for exact adherence to the time slots. Thus, data must be available on time at the sender and needs a high *reactivity* for fast processing at the receiver. For further improving the localization frequency, distance measurement and data aggregation tasks are both executed while the position estimation task still processes the information from the last measurement *simultaneously*. Thereby, the significant CPU load needs sophisticated *scheduling* and must provide *preemption* to avoid long-term blocking of highly reactive tasks. These real-time requirements, however, vary according to each individual task's current conditions. E.g. during distance measurement, the corresponding task temporarily needs a high priority. Beyond, other tasks should be privileged by means of *dynamic priorities*.

The just described tracking system was developed completely independent from the steering control and remote maintenance systems. Still, their tasks require interaction and *inter-task-communication*. Additionally, they must be composable even in case of potentially overlapping resource requirements. E.g. the motor control and measurement tasks share a common hardware timer to generate a preferably uninterrupted PWM signal for speed control and a sporadic square wave for driving the ultrasound transmitter, respectively. Finally, the fuzzy logic also produces high CPU load in parallel to the localization process and the remote maintenance system shall always be ready for external commands.

Next, we will outline problems of existing operating systems and our solution strategies regarding real-time operation.

III. Problems of current Systems and our Solution Strategies

Many available operating systems for sensor networks lack special support for such complex real-world scenarios. They often do not support preemptive or prioritized tasks, which hardly allows dynamic adaption to changing system requirements at runtime. In particular, the consequently priority unaware resource management policy further reduces the desired reactivity of time critical tasks. Some systems like TinyOS [1] and Contiki [2] follow an event-driven approach, where processes are implemented as event handlers that run to completion. Problematic in purely event-driven implementations is, that a lengthy computation completely monopolizes the CPU, which makes the system temporarily unable to quickly respond to external events. Additionally, it prohibits the interleaved execution of several computations. One solution to these problems is to provide a multi-threaded operation model. Although multi-threading extensions exist for some event-driven systems, they often lack important features like priorities or intelligent scheduling strategies (e.g. TOSThreads in TinyOS or Protothreads in Contiki). The basic architecture of these inherently non-preemptive systems also often limits the applicability of the multi-threaded approach and therewith the expected advantages.

Systems which inherently offer preemptive multi-threading are e.g. Mantis [3] and Nano-RK [4]. Nano-RK lacks dynamic priorites and uses the preemptive Priority Ceiling Protocol emulation Highest Locker Priority for resource assignment. This scheduling policy has the disadvantage that low prioritized tasks can block intermediate prioritized ones though the high priority task that defined the ceiling priority not even currently demands the resource in question. In fact, this leads to unnecessarily bad reactivity and can cause late results of the intermediate prioritized tasks. Mantis uses preemptive time-sliced priority based thread-scheduling with round-robin semantics within a priority level. Indeed, it provides no mechanism to limit priority inversion [5]. Although both operating systems provide a local system time, they do not offer the possibility to wait on events or resources with a limited timeout for reacting on various runtime imponderabilities. Finally, the low resolution of $1\ \mathrm{ms}$ prohibits precise event capturing as already required for various measurements like ultrasonic position estimation.

To meet the special requirements regarding reactivity and modularity of complex WSN/SANET applications, we developed the *Smart*OS real-time kernel to integrate four basic concepts. Most central is the inherent system timeline with a resolution of $1\ \mu\mathrm{s}$. It grants an individual local time for each node and is used for automatic timestamping of external events. Furthermore, it allows periodic tasks and limited waiting for events and resources. By definition, tasks are fully preemptive and possess a variable priority, each. This provides dynamic adjustment of their individual reactivity to changing system requirements and environmental conditions. The dynamic (de)allocation of resources at runtime allows tasks to use them just on demand and is essential for free composition of tasks to a more complex total system. While the priority inheritance protocol speeds up the resource handover from lower to higher prioritized tasks, we also implemented a mechansim of task cooperation which improves reactivity and even grants deadlock detection and recovery at runtime. Finally, an event concept is available for task interaction and synchronization. These events can be triggered by either tasks or external occasions.

In spite of its complexity regarding scheduling, task-interaction, time measurement and resource management, the presented application was successfully implemented on a single SNOW^5 [6] sensor node (MSP430@8MHz, 10KB RAM, 48KB ROM). With it, we achieved a localization frequency of approximately $6\ \mathrm{Hz}$ at a precision of around $2\ \mathrm{cm}$ and a track deviation of about $20\ \mathrm{cm}$.

IV. Conclusion

We showed, that real-time can already be relevant for today's WSN/SANET applications. Especially when task-parallelism and intense environmental interactions are required, available operating systems commonly lack special support. For enabling even resource constrained sensor nodes to execute complex real-time applications, we developed the embedded operating system *Smart*OS. Within $1.8\ \mathrm{kB}$ of ROM, *Smart*OS offers concepts which allowed us to successfully implement a complex vehicle tracking and steering application within a WSN. Therewith even difficult real-time problems can be mastered by using optimized software. For the future we expect that more powerful sensor nodes will enlarge the realization of even more ambitious applications. The design of *Smart*OS with its preemptive multi-threaded approach and its optimized scheduling and resource management policies suits a new broad class of applications which sensor networks already are or soon will be able to execute.

References

[1] U. Berkeley, "TinyOS," Web site http://www.tinyos.net/, UC Berkeley, 2004. [Online]. Available: http://www.tinyos.net/

[2] Adam Dunkels, Bjorn Gronvall, and Thiemo Voigt, "Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors," in *LCN '04: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 455–462.

[3] S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, and R. Han, "MANTIS OS: An embedded multithreaded operating system for wireless micro sensor platforms," *ACM/Kluwer Mobile Networks and Applications (MONET), Special Issue on Wireless Sensor Networks*, vol. 10, no. 4, pp. 563–579, August 2005.

[4] A. Eswaran, A. Rowe, and R. Rajkumar, "Nano-RK: An energy-aware resource-centric RTOS for sensor networks," in *RTSS '05: Proceedings of the 26th IEEE International Real-Time Systems Symposium*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 256–265.

[5] O. Babaoglu, K. Marzullo, and F. Schneider, "A Formalization of Priority Inversion," 1993. [Online]. Available: citeseer.ist.psu.edu/babaoglu93formalization.html

[6] M. Baunach, R. Kolla, and C. Mühlberger, "SNoW^5: A versatile ultra low power modular node for wireless ad hoc sensor networking," in *5. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze*, P. J. Marrón, Ed. Stuttgart: Institut für Parallele und Verteilte Systeme, 17.–18. Jul. 2006, pp. 55–59.