

Toward Easily Analyzable Sensor Networks via Structuring of Time-Triggered Tasks

Kane Kim and Yuqing Li

Dream Lab., UCI and
UCSD-UCI Cal(IT)², Irvine, CA
{khkim, yuqing}@uci.edu

Abstract

Software techniques which enable efficient construction of sensor networks that are easily analyzable and can rarely run into a chaotic situation are in high demands. With the motivation for developing such software techniques, a skeleton of a software architecture for sensor networks which uses time-triggered functions (TTFs) extensively has been formulated. The TTF is a fundamental building-block for software which realizes global time based coordination of distributed actions (TCoDA). TDMA communication is adopted and harmonious joining of a sensor node into an existing cooperating group of sensor nodes is facilitated. Approaches for supporting preemptive TTFs by use of a multi-threading OS kernel as well as the approaches for supporting non-preemptive TTFs by use of application-level components are also presented. The latter approaches were devised for the situations where incentives for avoiding the use of a multi-threading OS would exist.

Index Terms: Sensor, network, real time, time-triggered, TTF, service function, TDMA, TCoDA, wireless, kernel, preemptive, analyzable

1. Introduction

Research and development of intelligent tiny sensors which consume low power and contain sensing, computation, communication components, has become a steadily growing field in recent years [Agr00, Aky02, Hil02, She01]. Due to their low costs, such sensors may be densely deployed inside or close to the object of interest, dynamically forming a cooperative sensor network. New potential applications of sensor networks are recognized continuously in military and commercial areas such as command-control, surveillance, environment monitoring, targeting system, etc.

In some large-scale sensor networks, especially

those where wireless communications are used, scarce resources are communication channels. For example, in the current version of the prototype intelligent sensor developed at UC Berkeley and called the Mica platform, there is only one frequency radio channel of 40Kb bandwidth used [Hil02, Lev02]. Therefore, shared uses of communication channels must be designed with a great deal of optimization efforts. Distributed sensors must use the shared channels in well coordinated manners while operating their sensing mechanisms largely autonomously. How the sensor nodes will be clustered or spread out geographically is not always predictable at design time. The time costs for collision detections and repeated competitions for accessing channels can thus be prohibitively high unless careful coordination which prevents excessive collisions is enforced among sensor nodes.

We feel that one cost-effective way for letting distributed sensors share communication channels is to enforce some kinds of TDMA (*time-division multiple access*) rules [Kop97]. Making distributed sensors follow TDMA can be viewed as an instance of applying a broadly applicable fundamental principle, *global time based coordination of distributed actions* (TCoDA). Coordination is realized by use of globally referenced time information, or for short, *global time*, rather than “last-minute” exchange of messages. One natural way to make sensors to use TDMA communication channels and at the same time operate their sensing mechanisms autonomously is to design their sensing activities to occur within carefully chosen time-windows. This means to realize sensing activities in the form of executing *time-triggered functions* (TTFs). In a sense, distributed sensing activities are also coordinated by use of global time.

Global time also plays a pivotal role in fusing data from distributed sensors. Data which are not stamped the origination times in a format of global time cannot be effectively fused together.

It is our thesis that effective and extensive application of the TCoDA principle leads to sensor networks that are "easily analyzable". Such sensor networks can rarely create chaos. The TTF is a fundamental building-block for software which realizes TCoDA. The purpose of this paper is two-fold. First, a skeleton of a software architecture for sensor networks which uses TTFs extensively is presented. TDMA communication is adopted and harmonious joining of a sensor node into an existing cooperating group of sensor nodes is facilitated. Secondly, approaches for supporting preemptive TTFs by use of a multi-threading OS kernel as well as the approaches for supporting non-preemptive TTFs by use of application-level components are also presented. We believe that these approaches are applicable to a broad range of sensor networks. A preliminary prototype implementation of the software architecture and the TTF support mechanisms on a network of Mica platforms is nearly completed at the time of writing this article.

2. Sensor network software architecture based on time-triggered functions

2.1 Global time based coordination of uses of shared communication channels: TDMA communication

Figure 1 illustrates a sensor network consisting of 7 active sensors and one "power node". The power node is usually a general-purpose server node with relatively powerful hardware components (e.g., CPU, memory, and disk), abundant energy sources, and strong connections (e.g., wired connections) to the application customers. The power node collects data from all sensor groups and sends useful information to the application customers. The 7 active sensors form 3 sensor groups. Each sensor group has a "master" node. The link between two sensors represents the reachability of radio transmission from one of the two to the other. The power node also makes the TDMA slot assignment for all sensor groups.

Figure 2 illustrates a TDMA slot assignment for the sensor network in Figure 1. For Group 1, the first slot is reserved for use by the power node and the 3rd, the 4th and the 5th slots are reserved for use by member nodes, 1, 2, and 4, respectively. The 6th and 7th slots are blocked out since members of Group 2, Node 3 and Node 5, use those slots and transmissions by those nodes can be heard by a member of Group 1, i.e., Node 4. The second slot is a special slot, "community's slot", and it is always reserved in all sensor groups for announcements by the sensor nodes which want to join certain sensor groups.

The TDMA slot assignment for Group 2 is similar. The first slot is blocked out since it is reserved for use by Node 7 in Group 3 and the transmissions by Node 7 can be heard by a member of Group 2, Node 5. The 5th slot is marked "G" indicating that the slot is used by a node in a neighbor group which functions as a "gate" for Group 2. A gate node (e.g., Node 4) can relay messages from a neighbor group (e.g., Group 2) to the power node and messages from the power node to the neighbor group. Both members of Group 2, Node 3 and Node 5, can directly hear Node 4 in Group 1. Similarly, the 7th slot in the TDMA slot arrangement for Group 3 is marked "G" since that slot is reserved for use by Node 5 in Group 2 which functions as a gate for Group 3.

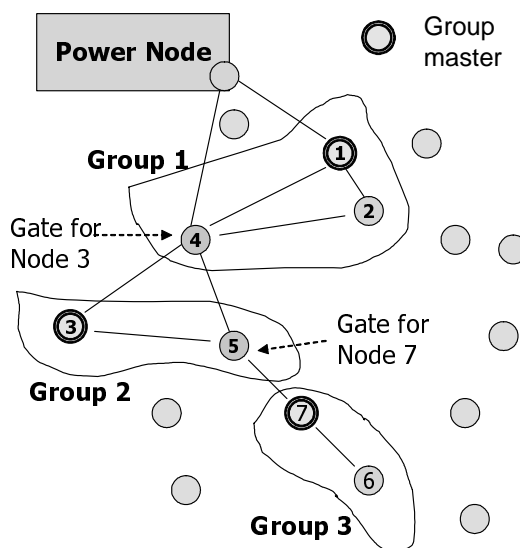


Figure 1. Sensor groups in a sensor network

Within a sensor group, a member can send a message to another normally *via the master of the group*. In other words, a master is a node which can communicate directly with all the members of the group. A member, which is not the master, is called a "worker" node. Therefore, worker nodes do not communicate directly among themselves.

TDMA Slot Assignment

Group 1	P	C	1	2	4	X ₍₄₎	X ₍₄₎	
Group 2	X ₍₅₎	C			G	3	5	
Group 3	7	C	6				G	

C := Community's slot for "Join" or special messages.

Figure 2. An example of a TDMA slot assignment for the sensor network in Figure 1

One obvious advantage of this TDMA approach is the minimal occurrences of collisions among sensor nodes in accessing wireless channels. For example, we are currently using in our laboratory Mica platforms which use a single frequency radio channel of 40Kb bandwidth. Due to a redundant coding scheme used, transmission of a 36-byte packet takes 24 milliseconds. Collision detection may take 24 milliseconds also. Thus it can be expensive. With the TDMA approach discussed above, collisions can occur during the community's slots but highly unlikely during other slots. Several sensors newly turned on may simultaneously attempt to announce their liveliness during the same community's slot. However, this does not create any critical situation for two reasons. First of all, newly born sensors are not yet parts of any operating sensor group and thus some delays in their joining into sensor groups can be easily tolerated. Secondly, it is possible to design sensor nodes to wait for different amounts of times after receiving announcements from authoritative nodes of *wake-up orders* before they attempt to announce their liveliness.

Without the approaches such as the above TDMA approach, attempts by a newly awakened node to join a cooperating sensor group may create significant interference with the on-going operation of the group. The global time base which is an essential ingredient in implementing the TDMA approach can be established in various ways nowadays. If GPS receivers are parts of the sensor nodes, then a global time base of microsecond-level precision can be established easily. Otherwise, a master-slave scheme which involves time announcements by the master as well as exploitation of the knowledge on the message delay between two nodes, can be used to establish a global time base of about 10 millisecond level precision with the intelligent sensors such as Mica platforms.

2.2 Global time based coordination of group merging and split

In sensor networks, neighboring sensor nodes are often organized as a group performing specific tasks. For example, each sensor node may periodically collect raw sensor values via one or multiple attached sensing mechanisms, perform basic signal processing on those raw data, and then extract relatively condensed information. Those condensed information are often exchanged within a group, and high-value information can be generated by combining and analysis of information in the group. As the last step, that high-value information produced by each group is propagated

via multiple gates to the most powerful computation unit who makes the final decision based on the information collected from wider areas. Efficient group formation and effective cooperation of group members are of fundamental importance in many sensor network applications.

In facilitating the joining of a newly awakened node into an existing cooperating sensor group, the following challenging goals must be met.

- (1) As a newly awakened node joins an existing cooperating group, the members of the group should be able to continue their application tasks without significant interruptions due to the joining of the new node.
- (2) Upon becoming awakened, a node should be able to join an existing group, receive an assignment of an application-specific task, and start performing its application task within an acceptable time bound.
- (3) As a member node crashes and thus loses its membership of a group, remaining members of the group should not experience any unacceptable interruptions in their application tasks.

In this section we will show how an efficient approach for facilitating the join of a node into a group and the timely cooperation of group members can be built on the basis of the TDMA assignment approach discussed in Section 2.1.

When a new sensor node S wakes up, it keeps listening to the communication channel until it hears an announcement message from a master node M twice. This announcement message is of course issued during a time-slot belonging to M and it contains among others an indication of when the next community's slot will arrive. S may be able to hear from more than one master. Then during the next community's slot, S sends out an announcement indicating its liveliness and the list of sensor nodes from which it has heard. That is, the announcement message sent by S includes its node ID and hearing abilities.

The attempt to make an announcement by S during the community's slot may run into collisions with other newly awakened sensor nodes. In such a case, S must wait for the amount of time chosen at design time before making another attempt. The retry should of course be made during a community's slot.

All master nodes must try to hear during the community's slots. Once a master node M receives a liveliness announcement from S , it submits a report containing the node ID and hearing abilities of S to the power node. The report may go through multiple gates before it arrives at the power node. Multiple master

nodes which have heard from *S* may submit reports to the power node.

Based on the report from *M*, the power node decides whether the newly awakened node *S* should merely become a member of the group led by *M* or a reorganization of several groups in proximity should occur. If the power node has received reports about *S* from multiple masters, the power node assigns *S* to an appropriate group. The decision also includes new TDMA slot assignments. The notice on the decision is sent to the relevant nodes including *M*. There are two cases where reorganization of multiple groups is necessary. First, if the hearing list of *S* includes sensor nodes belonging to multiple groups, the TDMA slot assignments for more than one group may have to be modified. Secondly, if the TDMA slots have become short in some sensor groups, the power node must reassign groups and their masters.

Once a master node received a new TDMA slot assignment for its group, it announces the new assignment to its members. Therefore, members of a group *must always listen during the slots belonging to their master*.

It is also possible that *S* may be located far from the power node and any other active sensor node although some sleeping sensor nodes may be nearby. Then *S* cannot hear any message. In such a case, *S* can be put into a power-saving sleeping mode after trying to hear for several TDMA cycles, and then later come back and try again.

The power node must always issue a message, containing at least a liveness signal and an indication of when the next community's slot will arrive, whenever its TDMA slot arrives. Therefore, when Node 1 in Figure 1 is the first sensor awakened, it can hear a message from the power node twice and then announce its liveness during the next community's slot. The power node declares that Node 1 is the master of the group containing Node 1 only, Group 1. The power node also gives a TDMA slot assignment for Group 1. When Node 2 wakes up, it can hear from Node 1 and announce its liveness during a community's slot. Node 1, the master, submits a report on Node 2 to the power node. The power node issues a new TDMA slot assignment for Group 1 which now consists of two nodes.

To continue the above illustration with Figure 1, assume that Node 4 wakes

up a little later and hears from the power node, Node 1, and Node 2. Node 4 then announces its liveness during a community's slot and this will be heard by the power node and the master of Group 1, Node 1. Node 1 will submit a report on Node 4 to the power node. Independent of this, the power node will announce a new TDMA slot assignment for Group 1 which now includes Node 4.

When Node 3 wakes up some time later, it can hear from Node 4 only. Moreover, when Node 3 announces its liveness, only Node 4 can hear it. Therefore, although Node 4 is not a master node, it must listen *during some community's slots*, at least once every few TDMA cycles. *All sensor nodes must do this*. When Node 4 sees the message from Node 3, the former realizes from the hearing list of the latter that it is the only node which can hear from the latter. Node 4 thus submits a report to the group master, Node 1, either during the slot belonging to itself (Node 4) or during a community's slot. Node 1 then forwards the report to the power node. (If Node 4 submitted a report during a community's slot, then the power node might have heard the report, too.) The power node announces the decision that Node 3 starts a new group, Group 2, and Node 4 serves as a gate for Group 2. The decision also includes new TDMA slot assignments for both Group 1 and Group 2. The decision notice reaches Node 3 through Node 1 and Node 4.

Therefore, joining of a newly awakened node into an established sensor group occurs in an "orderly" manner under the scheme discussed above. Here the case where all sensors use the same single frequency radio channel for communication has been dealt with.

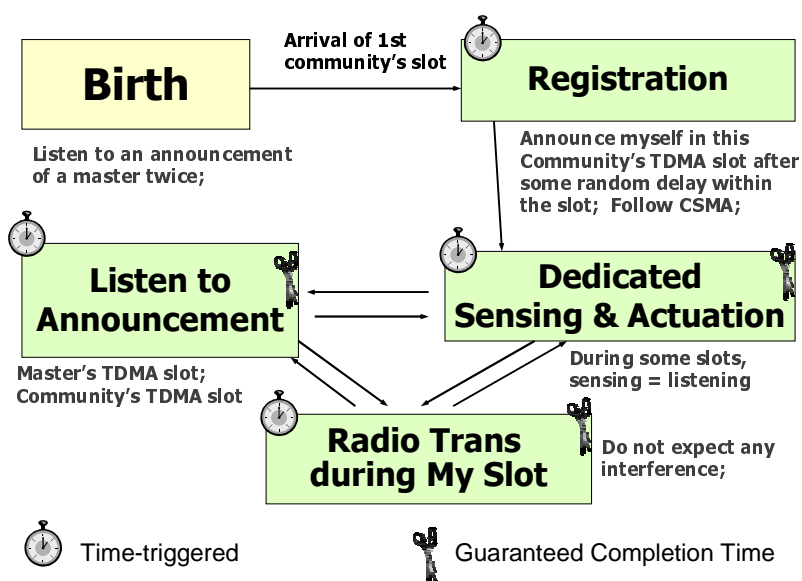


Figure 3. Operational phases of a sensor node

This orderly joining scheme can be extended easily to fit the case where multiple frequency channels are used. This joining scheme relies on the TDMA communication approach which is an instance of applying the TCoDA principle to communication channels. Cooperation among the power node, group masters, gates, and group members follows the TCoDA principle at least indirectly. Cooperative actions of each sensor node can be structured in natural forms of TTFs as will be shown in Section 3.

2.3 Global time based coordination of sensing activities and concurrent operation of multiple sensing mechanisms

Each member of a sensor group must operate one or more sensing mechanisms during the time-slots during which it does not have to pay attention to communication channels. Figures 3 and 4 depict this aspect. Each of the three phases in Figure 4 is time-triggered. The "dedicated sensing and actuation" phase in the figures can be further delineated into multiple more detailed states and complex transitions among the states depending on the nature of the applications.

Observing the TDMA rule requires operating an interval timer which generates interrupts. Also, message transmission and receiving over wireless channels often require enabling interrupts from relevant devices.

In addition, certain sensing mechanisms perform measurement activities for a while and then generate interrupts. Moreover, in general, multiple sensing mechanisms are in action concurrently. Most of the interrupts require time-bounded response. Therefore, it is of great importance to enable only the necessary set of interrupt generators and ensure by analysis that every possible interrupt can be handled within an acceptable time bound. It is highly useful to let the programmer specify explicitly along with each TTF the *set of interrupts to be enabled*.

3. An approach for supporting time-triggered functions

Facilitating easy use of TTFs in desktop computer or even powerful PDA environments equipped with preemption-supporting multi-threading operating system (OS) kernels has been studied to a considerable extent [Kim98, Kim02]. However, with the current hardware economy and miniaturization capabilities, industry is sometimes forced to avoid the use of multi-threading OS

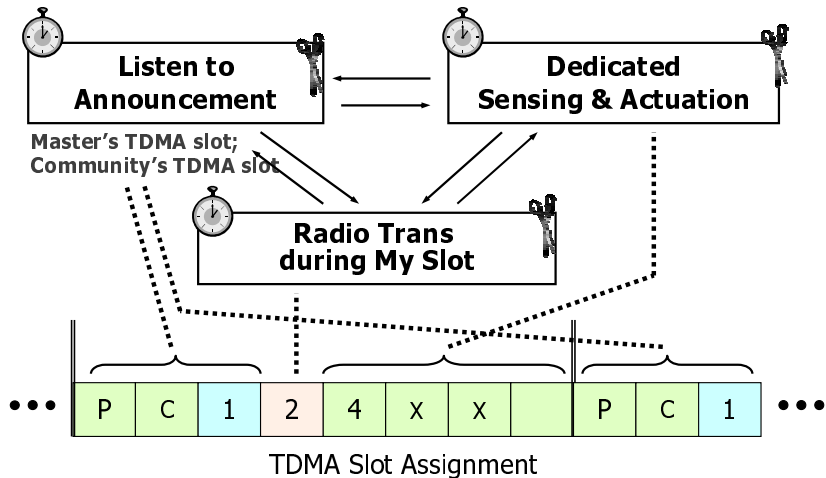


Figure 4. TDMA slots used in different operational phases of a sensor node

kernels for tiny intelligent sensors. This is due to the hardware limitations of certain sensors. For example, multi-threading involves memory commitment for thread contexts and stacks, which can be stressful to sensor programmers. TinyOS under development at UC Berkeley is a representative of non-preemptive OSs, under which the main function performs the ordering the tasks created or requested and after dispatching a task it takes control back only when the dispatched task is finished. All tasks must be reasonably short since a big time-consuming task can potentially block the executions of later ordered urgent tasks for a long time. However, when an intelligent sensor is equipped with multiple sensing mechanisms to be operated in parallel along with sizable memory, the costs of using a multi-threading OS kernel may be well justified.

3.1 Non-preemptive TTF support

The most general form of a TTF can be expressed as,

"From GlobalTime = T1 to T2, Do an execution of the TTF body Every P time-units (= iteration-interval) By GCT"

where GCT denotes the *guaranteed completion time* [Kim00]. TTFs are suitable for implementing periodic tasks which are often needed in sensor reading, communication, etc.

In order to establish a reasonably versatile programming environment, support for TTFs must be accompanied by the support for *service functions* (SvFs) which are complementary to TTFs. Event-triggered tasks are frequently encountered in sensor network applications, e.g., time-consuming functions needed only after certain interrupts occur, etc. We can model those tasks as SvFs. For example, an interrupt service routine

which runs in an interrupt-disabled mode can make a reservation for execution of an SvF performing sizable computation and then return the control back immediately to increase the concurrency of the system. Once a request or reservation for an SvF execution is recognized by the scheduling authority (e.g., *main()* function), the SvF execution task will be dispatched at the earliest appropriate time, e.g., the time at which the initiation of the SvF execution will not lead to conflicts with timely execution of any TTF. Under normal circumstances the SvF execution must be completed within the *maximum execution duration* (MED) determined at the design time.

SvF executions can also be requested by remote nodes. For example, a TTF in a sensor node can interact with a remote sensor node via a remote service call. It can be done when the delay caused by the preparation and issuance of the request in the local node running the TTF, the transmission of the request and the service result, and the processing of the service request in the remote node, is known to be bounded within an acceptable range. In such a case, a request for an SvF execution may take the form of an application message.

Figure 5 depicts an approach for implementing the support for non-preemptive TTFs. It was devised for the situations where incentives for avoiding the use of a multi-threading OS kernel would exist. The scheduling authority is a dedicated component, called the *TTF Coordinator*, and it handles the registration and scheduling of TTFs. The TTF Coordinator contains data structures that maintain *timing requirement specifications* for TTFs and SvFs, the *execution schedule* for TTFs and SvFs, and possible *error reports*. It also provides several interfaces for manipulation of those data structures as shown in Figure 6.

Application components contain bodies of TTFs and they register the TTFs with the TTF Coordinator along with relevant timing requirement specifications (e.g., loop start time, loop end time, iteration interval, earliest start time, latest start time, and guaranteed completion time) via API *RegisterTTF()*. The information registered may also include the specifications of *interrupts to be enabled*. Before accepting the registration of a new TTF, the TTF Coordinator performs a basic scheduling feasibility check, and if the feasibility check produces a negative result, the TTF Coordinator discards the TTF registration

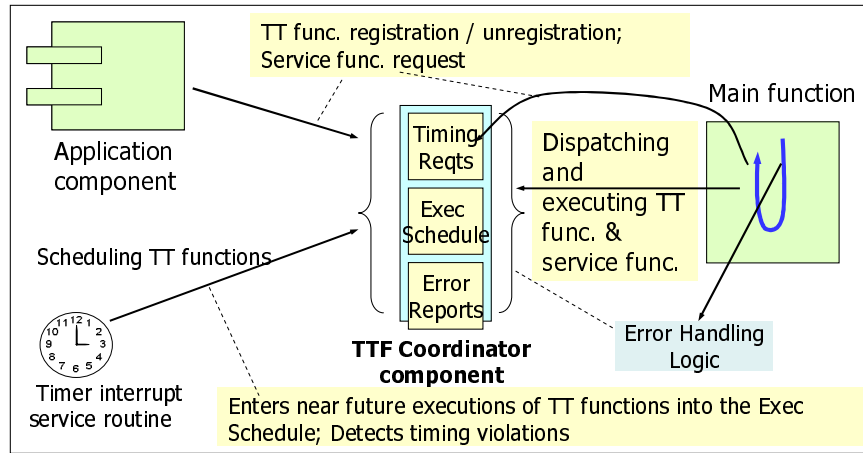


Figure 5. A scheme for implementation of the support for TT functions

request and records related error information.

The TTF Coordinator is responsible for maintaining the execution schedule by reflecting the timing requirements of registered TTFs and the SvF requests generated by application components. The execution schedule contains a queue of TTF execution plans sorted by their earliest start times and a queue of SvF requests sorted by their generation times. The two queues are modified when any of the following events occurs.

- (1) A TTF or SvF is dispatched, usually upon termination of the previously dispatched function.
- (2) A new TTF is registered.
- (3) An SvF request is generated.

```

Interface TTF-Coordinator
{
    /* to register a TT function with the coordinator */
    INDEX RegisterTTF (TTFRequest *);
    /* to unregister a TTF with the coordinator */
    Bool UnregisterTTF (INDEX);
    /* to invoke a service function */
    void InvokeSvF (SvFRequest *);
    /* to update the execution schedule based on */
    /* existing registered TTFs */
    void UpdateExecSchedule ();
    /* to dispatch next ready function in */
    /* the execution engine */
    FuncType* DispatchNextFunc ();
    /* to retrieve a timing error report */
    TimingErrorRept * RetrieveNextErrorRept ();
}

```

Figure 6. Interface of the TTF Coordinator component

(4) The timer interrupt service routine looks ahead and generates near future execution plans for registered TTFs.

The *main()* function dispatches the next TTF or SvF execution based on the execution schedule via API *DispatchNextFunc()*. It first checks the queue of TTF execution plans and identifies the TTF execution plans of which the earliest start times are already past. If multiple such TTF execution plans are found, the TTF execution plan of which the GCT (guaranteed completion time) is the earliest is dispatched. A dispatch here means just invoking the selected TTF. If there is no ready TTF available, i.e., if the earliest start time of the first TTF execution plan has not arrived yet, then a check is made whether the first SvF request in the queue of SvF requests can be fully honored, i.e., the corresponding SvF can be fully executed, before the latest start time of any TTF execution plan in the queue. If the result of the check is positive, that SvF is dispatched. Otherwise, no dispatch occurs.

The rule described above and enforced in dispatching or delaying an SvF was called the *basic concurrency constraint* (BCC) in [Kim97, Kim00]. Enforcing the BCC leads to significant reduction in the complexities of analyzing the achievable GCTs of TTFs. It eases the analysis of achievable GCTs of SvFs as well although this analysis is in general much more complicated than that for TTFs.

If the global time is already past the latest start time of a certain TTF execution plan in the queue, the *DispatchNextFunc()* function records error information but does not discard the TTF execution plan since missing the start time-window is normally not a critical error and only missing the output time-window or the GCT is a critical error. Therefore, such a TTF execution will still be dispatched. Nevertheless, it should be taken as a sign that the system design is not a sufficiently reliable one.

The *main()* function dispatches the selected TTF or SvF after enabling and disabling interrupts as specified. Once a function is dispatched, its execution cannot be preempted until its completion.

Again, in the scheme presented above, one can see that TTF executions are to be interfered by SvF executions minimally, if there is any interference. This arrangement was adopted for the ease of ensuring timely executions of TTFs. A side benefit is that even if there is data sharing between a TTF and an SvF, no run-time races occur.

Attempts to detect possible error conditions, such as deadline violations, can be made whenever the TTF Coordinator is accessed by the *main()* function, the timer

interrupt service routine, or application components. Once an error is detected, a corresponding report will be stored inside the TTF Coordinator. Application components or the *main()* function may retrieve the error reports later via *RetrieveNextErrorRept()* and take appropriate actions.

Since both the *main()* function and the timer interrupt service routine can access the same set of data structures in the TTF Coordinator and since their executions can be arbitrarily intertwined, all relevant accesses to the TTF Coordinator must be implemented as critical sections. A prototype implementation of the structure in Figure 5 which was based on Mica platforms is in the testing phase in the authors' laboratory.

3.2 Preemptive TTF support

In principle, many data structures, mechanisms, and interfaces devised for non-preemptive TTF support can be applied for preemptive TTF support. However, preemptive scheduling requires the provision of a multi-threading OS kernel. Then it is natural to put the TTF Coordinator inside such a kernel, i.e., at a layer below the application layer. The dispatch function is also naturally provided by the kernel unlike the situation for non-preemptive TTF support where the *main()* function can handle dispatching. Such OS mechanisms including kernel and middleware mechanisms supporting TTFs have been studied in recent years [Kim98, Kim02].

4. Conclusion

In this paper, we have presented a skeleton of a software architecture for sensor networks which uses TTFs extensively. Our interests are in realizing sensor networks that are "easily analyzable". We believe that such sensor networks can be obtained through effective and extensive application of the TCoDA principle. The TTF is a fundamental building-block for software which realizes TCoDA.

We have also presented approaches for supporting non-preemptive TTFs by use of application-level components as well as those for preemptive TTFs by use of multi-threading OS kernels. Non-preemptive TTFs and their implementation approaches were devised for the situations where incentives for avoiding the use of a multi-threading OS kernel would exist. We believe that all these approaches are applicable to a broad range of sensor networks. A prototype implementation of the software architecture and the TTF support mechanisms on networks of Mica platforms is in the testing phase in our laboratory.

Preemptive TTFs can be used for achieving better

performance but supporting them requires multi-threading OSs which contribute to increased memory requirements. In the long run, with larger-capacity hardware available in future sensor network systems, the TTF support facility can be easily extended to support high-level real-time distributed object programming models such as the Time-triggered Message-triggered Object (TMO) in which both TTFs and SvFs take the forms of object methods [Kim97, Kim00, Kim02].

The software architecture discussed in this paper is a young research subject. Much more research is needed on aspects such as timeliness guarantees, message routing, fault tolerance through redundancy and reconfiguration, etc.

Acknowledgements: The research work reported here was supported in part by the NSF under Grant Numbers 02-04050 (NGS) and 00-86147 (ITR), and in part by the US DARPA under Contract F33615-01-C-1902 monitored by AFRL. No part of this paper represents the views and opinions of any of the sponsors mentioned above.

References

[Agr00] Agre, J. and Clare, L., "An integrated architecture for cooperative sensing networks", *IEEE Computer*, Vol. 33, Issue 5, May 2000, pp.106-108.

[Aky02] Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., and Cayirci, E., "A survey on sensor networks", *IEEE Communications Magazine*, Vol. 40, Issue 8, Aug. 2002.

[Hil02] Hill, J.L. and Culler, D.E., "Mica: a wireless platform for deeply embedded networks", *IEEE Micro*, Vol. 22, Issue 6, Nov/Dec 2002, pp.12-24.

[Kim97] Kim, K.H., "Object Structures for Real-Time Systems and Simulators", *IEEE Computer*, August 1997, pp.62-70.

[Kim98] Kim, K.H. and Subbaraman, C., "Principles of Constructing a Timeliness-Guaranteed Kernel and the Time-triggered Message-triggered Object Support Mechanism", *Proc. ISORC '98 (IEEE CS 1st Int'l Symp. on Object-oriented Real-time distributed Computing)*, Kyoto, Japan, April 1998, pp. 80-89.

[Kim00] Kim, K.H., "APIs for Real-Time Distributed Object Programming", *IEEE Computer*, June 2000, pp.72-80.

[Kim02] Kim, K.H., "Commanding and reactive control of peripherals in the TMO programming scheme", *Proc. ISORC '02 (IEEE CS 5th Int'l Symp. on Object-Oriented Real-Time distributed Computing)*, Crystal City, VA,

April 2002, pp.448-456.

[Kop97] Kopetz, Hermann, '*Real-Time Systems: Design Principles for Distributed Embedded Applications*', Kluwer Academic Publishers, 1997.

[Lev02] Levis, P., and Culler, D., "Mate: A Tiny Virtual Machine for Sensor Networks", *ASPLOS*, Dec. 2002.

[She01] Shen, C.C., Srisathapornphat, C., and Jaikaeo, C., "Sensor information networking architecture and applications", *IEEE Personal Communications*, Vol. 8, Issue 4, Aug. 2001, pp.52-59.