Composable Component Interfaces for Time-Triggered Systems

Peter Puschner and Bernhard Frömel
Technische Universität Wien, 1040 Wien, Austria
Email: {peter,froemel}@vmars.tuwien.ac.at

***Abstract*—When building real-time systems out of networked components, the use of temporal-firewall interfaces between the components supports a composable system design. This paper shows that, depending on the requirements of an application, the temporal-firewall interfaces of a time-triggered communication system can be accessed in two ways, either as data-sharing interfaces with asynchronous access or as time-aware interfaces with time-synchronized access. The paper describes each of these two ways of interfacing with a time-triggered communication system and characterizes their differences.**

I. Introduction

Computer systems that control safety- and time-critical embedded applications (e.g., medical devices, cars, powerplants) need more and more computational power and therefore get larger in terms of number of components (e.g., nodes of a distributed system). We have to make sure that this growth in size and the additional interaction needs of the components caused by this growth do not create interferences that increase a system's complexity, thus infringing the safety of the applications. It is therefore important that the components of such multi-component systems act highly autonomously, i.e., the point of control that determines *which* actions a component performs and *when* these actions are triggered should reside *within*, not outside the component. Component autonomy ensures that the subsystems can be developed independently and their functional and temporal properties can be verified and validated in isolation, thus cleanly separating the responsibilies of the suppliers of the different subsystems [1].

Prior work has shown that time-triggered communication supports the construction of multi-component systems that allow components to keep their autonomy in the time domain [1]. Time-triggered communication does not impose control pressure on receivers as data objects exchanged via the *temporal-firewall interface* [2] of the communication system have the semantics of variables that are updated by state messages. Unlike event-triggered communication systems, time-triggered communication systems do not use interrupts (i.e., signals that exert external control on the receiving component) to notify components about message arrival. Further, state-message communication does not impose variable external load on receiving components.

Time-triggered communication eliminates the main source for external temporal control on component behavior. Still, the data-sharing interface between the computing unit and the communication controller of a component needs to be designed with care, in order to take full advantage of the temporal-firewall interface (i.e., to eliminate control interferences). In this paper we discuss two possible ways in which autonomous subsystems can access temporal-firewall interfaces in a time-predictable way. The first approach allows for a low build complexity of the subsystems – subsystems use the temporal-firewall interface as an *asynchronous* interface that masks potential control conflicts (see Section IV-A). In the second approach, subsystems are *time-aware* and *synchronize* to the global timebase of the time-triggered communication service. They use information about the send and receive times of messages to avoid control conflicts when accessing the interface and to obtain a consistent view of the global system state. This approach is for fault-tolerant systems that need to meet tight timing constraints (see Section IV-B). After introducing each of the two interfacing methods, Section V compares their contrasting properties.

II. Time-Triggered System Model

A *Time-Triggered System (TTS)* is a distributed real-time computer system adhering to the principles of the Time-Triggered Architecture (TTA) [1], see Figure 1. It consists of a *Time-Triggered Communication Subsystem (TTCSS)* and interacting, autonomous *Computational Components (CCs)*.

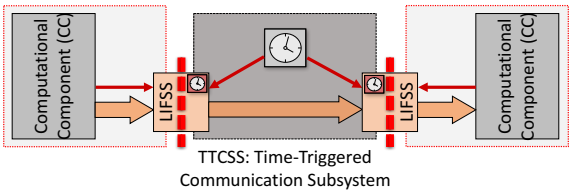

Fig. 1. Time-Triggered System Model.

The TTS provides a service to its *time-sensitive environment*. This involves the precise temporal coordination of output actions that are controlled by the CCs. To support the temporal coordination of the CCs, a TTS maintains a *global clock* by synchronizing the local digital clocks of the CCs to a precision better than the precision bound Π. The precision bound Π is determined by the dynamics of the environment.

A. Real-Time Data-Flows as Periodic State Messages

A conflict-free *TTS-wide communication schedule* specifies all message send- and receive operations with respect to sender, receiver(s), and message size. Following this communication schedule, a TTS transmits *periodic state messages*

with defined periods. A state message contains an observed state (e.g., humidity of a room) at a specific instant.

B. Linking Interface Subsystem

The Linking Interface Subsystem (LIFSS) of a CC contains the functionality for accessing the global timebase and the Time-Triggered Communication Subsystem (TTCSS). It performs local clock synchronization, provides the global time to applications executed by its CC, and provides means to exchange messages with other CCs.

For message exchange, the LIFSS has a *shared memory* that is partitioned into *channel endpoints*, called *ports*. In receive operations, the TTCSS writes message data to these ports and the CC reads them. For send operations, the CC writes data to ports and the TTCSS reads them and transports messages to their destination(s).

The global timebase drives the *time-triggered control engine* of a TTS. The time-triggered control engine ensures that messages are sent to and received from the TTCSS at the send or receive instants specified in the TTS-wide communication schedule. Further, the time-triggered control engine can be configured to notify the CC about completed send- and receive operations by triggering an interrupt or setting status flags. Finally, the time-triggered control engine allows for setting up (periodic) global clock-state notifications, i.e., periodic clock interrupts. CCs can use these clock interrupts to synchronize their operation to the global clock.

C. Time-Triggered Communication Subsystem

The Time-Triggered Communication Subsystem (TTCSS) realizes encapsulated, time-deterministic and reliable message channels between CCs. It transports messages of a defined size from a sender CC to one or more receiver CCs with a predictable delay.

III. Interfaces and Control Flow

In a generic component-based system, components communicate by the exchange of messages via their interfaces. Besides the data flow of messages, it is important to examine the *control flow* among interacting components. The control flow determines who initiates message transfers and is in command of the interaction. Let us consider the transfer of a message from a sending to a receiving component. If the sender is in control of transferring the message, then the control flow originates from the sender and terminates at the receiver. We call this an *information-push* [1] operation. If the receiver is in control of transferring the message from a sender, then control flow and data flow are in opposing directions. We call this an *information-pull* [1] operation.

Information-push operations are ideal for senders. For an information-push operation, the sender does not have to wait until the receiver is ready, nor does it need to reserve resources for message buffering while waiting. Information-pull operations are ideal for receivers, to process messages under their own control. In particular, message-pull receivers cannot be disturbed by message arrivals occurring at times that are not under their own control.

In a TTCSS, there is no explicit control flow across the communication channels. Due to the program-variable semantics of state messages, message arrival does not impose external control on CCs. So, CCs may, in principle, read ports in information-pull operations and write to ports in informaion-push operations. The only potential control conflicts at a LIFSS can arise due to mutual-exclusion requirements of accesses. We will show how the mutual-exclusion control conflicts can be resolved by taking advantage of the properties of time-triggered communication resp. by using LIFSS services.

IV. Temporal-Firewall Interfaces

In the following we will discuss the two alternative interfacing strategies that CCs may adopt at their temporal-firewall interfaces. The chosen interfacing strategy will determine how and when CCs use the ports and the interrupt of their LIFSSs.

The components of a TTS may use the LIFSS either as a

- time-agnostic, asynchronous interface, or
- time-aware, time-synchronized interface.

Which strategy system designers will choose for a particular application will depend on the application requirements.

A. Time-Agnostic, Asynchronous Interface

From the point of view of control autonomy, it is easiest if a component uses the LIFSS as a time-agonostic, asynchronous data-sharing interface. The CC writes to or reads from this interface whenever there is a need to do so.

On asynchronous interface access, the real-time component is agnostic about the current message transfers on the time-triggered communication medium. The task schedule on the CC and the message schedule of the TTS are independent. To schedule its tasks, the CC may use any single-processor real-time scheduling technique that suits the needs of its application software.

1) Accessing the LIFSS: A time-agnostic interface does not restrict when readers or writers access the interface, i.e., it allows for simultaneous LIFSS accesses by a CC and the TTCSS. We therefore have to ensure that the mechanisms that warrant the data consistency of simultaneous LIFSS accesses do not affect the timing of either of these subsystems.

To avoid timing disturbances by simultaneous accesses to the LIFSS, we propose to use a variant of the Non-Blocking Write Protocol (NBW) [3] for interface-read or write operations (see Figure 2). The write operation is identical to the write of NBW. It runs the same sequence of instructions every time it executes. Our read operation differs from the original NBW read in that our version always executes the same instruction sequence. The original version follows different instruction traces depending on whether an access conflict to the LIFSS occurrs or not; this leads to different control flows and variable timing in the reading subsystem.

The here-presented version of the interface-read operation uses single-path code [4], [5] generated by a specific compiler.

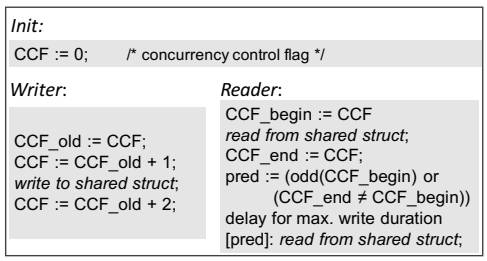

Fig. 2. Variant of NBW.

This compiler replaces branches in the control flow by predicated execution (see last code line of the reader). In predicated execution, the sequence of executed instructions is always the same — instruction predicates instead of branches control how the code manipulates data.

If executed on time-predictable hardware [6], not only the instruction sequence, but also the timing of single-path code is invariable. This timing invariability masks conflicting LIFSS accesses and prevents the interface from affecting the timing or control behavior of CCs resp. TTCSSs.

2) Interface Timing Properties: There are two questions of interest when tranferring real-time data via the interface: (i) What information about real-time observations do we have to transmit via the interface? (ii) What message-transport jitter do we have to expect at the interface?

An observation is characterized by a *(name, value, time)* tuple [1]. When transporting an observation via the interface, the observation *name* is implicitely coded by the address we write to resp. read from at the interface. *Value* and *time*, however, have to be explicitely handed over via the LIFSS.

For an asynchronous interface, the jitter of the data transfer of an observation depends on the transmission period of the data item by the TTCSS. As component-write operations to the interface are not synchronized with the transmission via the TTCSS, the jitter of the transmission of data after a component-write operation equals the period of the message that transports the data. Analogously, the jitter of the interval between the data arrival at the LIFSS and the read operation by a CC equals the message period. Adding up the jitter at both ends yields a total transmission jitter of twice the period of the message that transports the data item.

B. Time-Aware Interface with Time-Synchronized Access

The time-synchronized access strategy avoids access conflicts to the LIFSS. To do so, a CC aligns the timing of its read and write operations to the LIFSS so that they do not coincide with LIFSS accesses of the TTCSS — the timing of the latter is derived from the TTCSS message schedule that is available at system-construction time.

1) Accessing the LIFSS: When designing the software for a CC with time-synchronized LIFSS access, one will pay particular attention to synchronize the time of read and write operations from the very beginning, thereby avoiding simultaneous accesses by the CC and the TTCSS. On the other hand, one can make best use of the timing information contained in the message schedule by scheduling all task activations on a CC tailored to the timing of LIFSS-read and write operations: In such a schedule, a task that reads data from the LIFSS will be activated shortly after the message with these data has been received and made available by the LIFSS. In a similar way, a task that writes data to the LIFSS will be scheduled in time to write its data to the LIFSS just before the TTCSS will transmit them.

To align LIFSS-read and write operations with the actual message transfers, the CC will need (a) a real-time task scheduler and (b) a mechanism for synchronizing its activities with the activities of the TTCSS. As mentioned above, the task schedules will have to avoid access conflicts to the LIFSS. This can, e.g., be achieved by using a static, table-driven scheduler, where the scheduling table guarantees that the LIFSS accesses of the CC are performed in time windows that are conflict-free. The programmable clock interrupt of the LIFSS is the means to synchronize the clock of the CC with the global timebase. This clock synchronization will keep the task scheduler of the CC and the message schedule of the TTCSS synchronized.

2) Interface Timing Properties: Synchronizing the LIFSS accesses of the CCs and the TTCSS has a positive effect on the timing properties of the interface.

First, synchronizing both LIFSS-write and read operations on all CCs to global time reduces the message-transport jitter in comparison to non-synchronized access. In fact, if we use a synchronized, table-driven scheduler to trigger the LIFSS reads and writes, the jitter can be kept as small as Π, the precision of the global clock.

Second, using table-driven schedules that are aligned to the global clock allows system designers to streamline task executions and message communication. This way, the overall response times of real-time transactions spreading over two or more CCs can be kept short.

Third, if all data manipulations and data transfers via the TTCSS follow a global schedule, then the information about the age of data items is implicitly available at all CCs of the computer systems. As a consequence, the time stamps of real-time observations do not have to be transported via the TTCSS. This means, in a time-triggered network with time-synchronized CCs, only the *values* of observations have to be handed over to the LIFSS. Both, the *name* and the *time* of the observation are implicitly known.

V. Comparison

Each of the presented access strategies to temporal-firewall interfaces maintains the automomy of CCs and makes components time-composable. The different interface-access strategies are, however, suited for distinct types of applications as they are paired with significant differences in the characteristics of the components that constituate the overall system. We therefore discuss the differences of the component characteristics in the following; see also Table I.

A component that accesses the LIFSS *asynchronously* acts fully autonomously, i.e., it ignores the message schedule and

TABLE I
COMPONENT CHARACTERISTICS FOR THE TWO INTERFACING METHODS.

Characteristic	*Interface*	
	Data Sharing	*Time Synchronized*
Control paradigm	full autonomy	adaptation to schedule
Message-transport jitter	2× msg. period	precision of clock
Task-msg. streamlining	no	yes
Use of time	value (explicit)	control (implicit)
Fault-tolerance support	no	yes
Build complexity	low	medium to high
Application examples	movie streaming, sensor network	engine control, film stretching

potential conflicts when interacting with the LIFSS. A *time-aware component*, in contrast, uses its knowledge about the message schedule to synchronize its LIFSS accesses to the operation of the TTCSS.

The fact that components with an asynchronous interface do not synchronize with the message schedule leads to a transport jitter of two times the message period for all data items read via the LIFSS. This jitter is significantly higher than for time-aware interfaces, which is Π, the precision of the global clock.

The lack of synchronization at asynchronous interfaces inhibits the streamlining (i.e., tight synchronization) of tasks that read from or write to the interface and the messages that transport the respective data items. A time-synchronized interface, in contrast, allows for the synchronization of these activities. Thus time-aware interfaces support real-time transactions with much shorter end-to-end delays than asynchronous interfaces.

When all operations of the components and the communication system are time-triggered, controlled by a global execution plan, then the knowledge about the points in time of all data-read and write operations are globally known, i.e., the timestamps of these operations are implicit global knowledge in the system. As a consequence, and in contrast to systems which operate asynchronously, one does not need to transport the times of real-time observations in synchronous systems.

Synchronizing the interface-read and write operations of a component with global time is a prerequisite for a clear definition of the state of the component, which, in turn, is needed for the construction of fault-tolerant real-time systems. Fault-tolerant real-time systems therefore require time-aware LIFSS access of all CCs.

The central advantage of using asynchronous interfaces is the low build complexity of the components and the overall system. Using asynchronous interfaces, developers do not need to know about the temporal particularities of the LIFSS or other components. In contrast, to fully exploit synchronized interface access, the timing of operations on the components and message transmission on the TTCSS have to be tightly coordinated. The complexity and cost for designing such a system-wide coordination may be significant.

Typical example applications for asynchronous interfaces are entertainment systems, like movie streaming, or monitoring systems (e.g., based on sensor networks). Usually, these applications do not require system-provided fault tolerance, but they greatly benefit from the lower build complexity. Still, they may need to compensate for jitter in the order of message periods during message reception by appropriate buffering or queuing. In the case of monitoring systems, observations can be time-stamped by reading the global time available at the LIFSS and explicitly attaching it to the observation. Applications that use time-synchronized interfaces trade the added build complexity for the realization of deterministic real-time transactions with short end-to-end delays. Typical examples of such applications are related to control (e.g., engine control systems, film stretching systems, ...), which often require highly regular sense-control-act cycles. Some control systems are safety critical and profit from fault tolerance that is enabled at the architectural level by the use of time-synchronized interfaces.

VI. CONCLUSION

In this paper we compared two strategies of accessing temporal-firewall interfaces. Temporal-firewall interfaces eliminate control interferences between communicating components and network interfaces; their use keeps the construction of composable real-time systems simple.

Temporal-firewall interfaces can be either used as asynchronous interfaces or as time-aware interfaces where components use a notion of global time to avoid conflicts when accessing the interface. Asynchronous interfaces are easier to implement. On the other hand, time-aware synchronized interface access has many advantages, ranging from a much smaller communication jitter, to a better synchronization of actions on different components, to the possibility of streamlining computation and communication activities, thus reducing overall transaction response times. Finally, synchronized interface access and operation support component replication for fault-tolerant systems.

ACKNOWLEDGMENT

This paper was partially funded by the EU COST Action IC1202: Timing Analysis on Code Level (TACLe). The research leading to these results has received funding from the European Union Seventh Framework Programme FP7/2007-2013 under grant agreement No 610535 - AMADEOS.

REFERENCES

[1] H. Kopetz, *Real-Time Systems*, 2nd ed. Springer, 2011.

[2] H. Kopetz and R. Nossal, "Temporal firewalls in large distributed real-time systems," in *Proc. 6th IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*. IEEE, 1997, pp. 310–315.

[3] H. Kopetz and J. Reisinger, "The non-blocking write protocol NBW: A solution to a real-time synchronisation problem," in *Proc. IEEE Real-Time Systems Symposium*. IEEE, 1993, pp. 131–137.

[4] P. Puschner, "The single-path approach towards wcet-analysable software," in *Proc. IEEE International Conference on Industrial Technology*, 2003, pp. 699–704.

[5] P. Puschner, R. Kirner, B. Huber, and D. Prokesch, "Compiling for time predictability," in *Proc. SAFECOMP 2012 Workshops (LNCS 7613)*. Springer, 2012, pp. 382–391.

[6] M. Schoeberl, S. Abbaspour, B. Akesson, N. C. Audsley, R. Capasso, J. Garside, K. Goossens, S. Goossens, S. Hansen, R. Heckmann, S. Hepp, B. Huber, A. Jordan, E. Kasapaki, J. Knoop, Y. Li, D. Prokesch, W. Puffitsch, P. P. Puschner, A. Rocha, C. Silva, J. Sparsø, and A. Tocchi, "T-CREST: time-predictable multi-core architecture for embedded systems," *Journal of Systems Architecture - Embedded Systems Design*, vol. 61, no. 9, pp. 449–471, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.sysarc.2015.04.002