SCHEDULING TOOL FOR RECONFIGURABLE TIME-TRIGGERED EMBEDDED SENSOR NETWORKS

Wilfried Elmenreich, Christian Paukovits, Stefan Pitzek

Institut für Technische Informatik
TU Vienna, Austria
{wil,pauko,pitzek}@vmars.tuwien.ac.at

ABSTRACT

Time-triggered systems are advantageous for embedded applications, where determinism and certification are required. However, when it comes to flexibility, time-triggered systems often require that possible extensions have been planned in advance, which makes it difficult to apply unforeseen changes to such a system. This paper presents an approach that overcomes this problem by an automatic configuration of time-triggered distributed applications. Each application is decomposed into services and mapped on a set of distributed nodes, whereas each node hosts one or more services. Each service is defined by the following interfaces: a real-time input/output, a configuration and planning, and a diagnostic and management interface. Services, nodes, the application, and the global system configuration are represented with XML-based description formats, that provide a language-neutral semantic specification of the respective properties. A scheduling tool uses these descriptions to generate a time-triggered application that complies to the application specification.

1. INTRODUCTION

Time-triggered systems have the great advantage over traditional event-triggered systems that they are easier to understand and to analyze [1]. Furthermore, the time-triggered paradigm supports composability with respect to the temporal behavior [2] of a system.

However, due to the static scheduling approach of time-triggered systems, event-triggered systems are more flexible than time-triggered ones [3].

One reason for this difference in flexibility is the higher off-line configuration effort required for time-triggered systems, since communication usually depends on schedules that must be calculated a priori. Accordingly, manual configuration becomes difficult or even infeasible very fast, so adequate tool support is especially important. Since time plays such a central role, the automatic creation of communication schedules is essential for automating configuration tasks for time-triggered systems. In this paper we present a scheduling tool that has been developed for this purpose. Since a scheduler cannot perform actual configuration tasks by itself, we also outline the overall system model and how the scheduling tool interacts with other parts of the system.

There exist approaches to increase flexibility in time-triggered systems. Almeida et al. present a flexible time-triggered communication approach that operates on a CAN network [4]. In this approach, messages are scheduled by a central authority, which allows for a compromise between flexibility and run-time overhead of the planning scheduler. Other approaches [5, 6, 7] follow an integration between a safety-critical static time-triggered communication channel with a flexible event-triggered channel. Lisner proposes a flexible TDMA slotting scheme, where communication partners have a fixed schedule in a regular part and may request extra slots in an extension part [8]. On the other hand, model-based approaches like Giotto [9] target the problem from a different side by separating the temporal properties of an application from its implementation. This facilitates the problem of modifying the static schedule, since the schedule is automatically created from the application specification by a re-run of the compiler tool.

In this paper we propose a model-based approach to support flexibility and composability for a time-triggered system. The distributed application is decomposed into services, where each service is described by its interfaces, i. e., a real-time input/output, a configuration and planning (CP), and a diagnostic and management (DM) interface. The overall application is defined by the interconnection of services, and specified with XML descriptions. A scheduling tool uses this formal description to generate the static schedule for the time-triggered application and verifies whether the schedule fulfills the temporal requirements or not. The unused bandwidth is used for diagnostic and management purposes. When a change or extension of the application is necessary, the XML descriptions are adapted and the tool recreates the time-triggered schedule and performs the verification again. If verification fails, the tool reports that the given resources (e. g., bandwidth) are not sufficient for creating a valid schedule.

The approach has been implemented on embedded hardware using TTP/A [10] as time-triggered communication protocol.

The remainder of the paper is organized as follows: Section 2 deals with the underlying conceptual model of this approach. Section 3 presents the scheduling tool and algorithm and Section 4 applies the tool in a case study. Section 5 concludes the paper and gives an outlook.

2. CONCEPTUAL MODEL

In the presented architecture, a distributed embedded application is described by a set of interconnected real-time *services*. A service is described by its operational interface. In embedded real-time systems we can distinguish the following interface classes:

Service Providing Linking Interface (SPLIF): This interface provides the real-time service results to other services (cf. [11]).

Service Requesting Linking Interface (SRLIF): A service that requires real-time input information requests these data via the SRLIF (cf. [11]).

Diagnostic and Management (DM): This interface is used to set parameters and to retrieve information about intermediate and debugging data, e. g., for the purpose of fault diagnosis. Access of the DM interface does not change the (a priori specified) timing behavior of the service.

Configuration and Planning (CP): This interface is used during the integration phase to generate the "glue" between the nearly autonomous services (e. g., communication schedules). The CP interface is not time critical.

A service may also have local interfaces to physical sensors, actuators, displays, and input devices, for which the service creates a consistent access via the SPLIF or SRLIF interfaces.

An application consists of one or more services that interact with each other via the real-time interfaces SPLIF and SRLIF. One or more services are implemented per *node*, whereas nodes are loosely coupled via a real-time communication system. Services that require local data exchange communicate via a shared memory interface. Services that are required to exchange data between different nodes communicate via the node's communication interface.

The representation of an application by its services only deals with the functional and data flow parts of the application. Additionally, we also require a temporal specification of a service that deals with the following properties:

- Service-specific properties that must be configured depending on the requirements of the application (e. g., the update rate of an actuator).

- End-to-end requirements of the application that can only be specified and verified at the level of the distributed application (e. g., end-to-end signal delay in control loops).

We use the so-called Interface File System (IFS) [12] as central interfacing mechanisms to the smart transducer nodes. The IFS is an "extended" shared memory that acts as a temporal firewall between smart transducer nodes and the network. The memory is organized as a simple flat file systems, where files store data and can be executed. All application and configuration related information is mapped to the IFS and thus available at the network level over the respective interfaces. Thus, it also hides internals of the node for complexity reduction.

The service-specific properties are modeled in the semantic description of the service. The end-to-end requirements of the application are expressed by so-called *dependencies*. We have identified the following kinds of dependencies:

Connection: This dependency represents the data flow between ports of services. A connection is directed by having a source and a target part. An input port may have only one connection to an output port, while an output may feed several input ports.

Causal: By defining a causal dependency between services, all in-between services (on the directed application graph) must comply to this dependency. In our context causal dependencies always incur timing requirements. An *instant* identifies the timing requirements of participating services. We distinguish the instant before and after execution. The *before* instant of a service is before the service task is executed, i. e., the moment when input data must have arrived. The *after* instant happens after the service task execution, thus a duration of WCET$_{TASK}$ after the before instant, where WCET$_{TASK}$ is the worst case execution time of the service task.

Phase: The phase dependency specifies non-causal time-related dependencies of instants among services.

All the presented properties are used to model the application requirements. All requirements are expressed in XML descriptions, which support interaction among tools (e. g., for modeling, code generation, and verification).

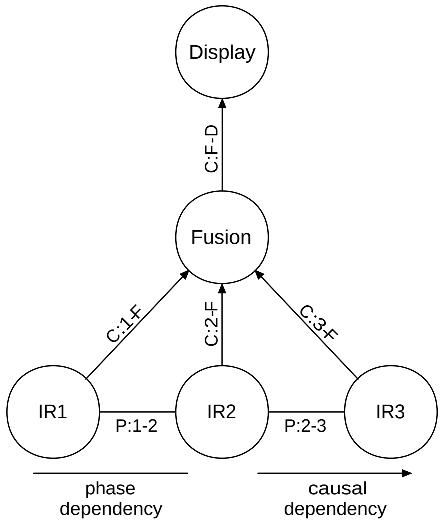

Fig. 1. the Precedence Graph of the smart fusion application

3. SCHEDULER FOR A TIME-TRIGGERED REAL-TIME SYSTEM

This section presents a scheduling and verification tool that uses the XML application descriptions that have been defined in [13].

3.1. Scheduling Algorithm

The operation of the TTP/A Scheduling Algorithm produces a Round Definition List (RODL) for each application in the cluster configuration description written in XML. [1] This algorithm had been implemented in a tool called the TTP/A Scheduler, which is described in detail in [14].

The main idea of the TTP/A Scheduling Algorithm is to represent the set of service tasks and their dependencies as a graph, the so-called Precedence Graph. Each *vertex* of the graph embodies an associated service task, whereas an *edge* represents a *causal dependency* between the bordering vertices, i. e., the service tasks. Additionally, (causal) edges are directed to represent the direction of the data flow and precedence of that dependency. An undirected edge in the Precedence Graph corresponds to a phase dependency. Figure 1 shows the Precedence Graph of the Smart Fusion Application [15].

We define three conditions that a valid Precedence Graph must satisfy:

- A service task must not depend on itself, neither directly nor indirectly. In terms of graph theory we can say, that the Precedence Graph must not include *loops* among any vertices.

[1]The RODL is the facility in TTP/A for specifying the timing of communication and task invocation.

$\mathcal{S}$	...	set of services
$\mathcal{K}$	...	set of causual dependencies
$\mathcal{C}$	...	set of connection dependencies
$\mathcal{P}$	...	set of phase dependencies
V	...	set of vertices in the Precedence Graph (PG)

```
while(there exist unmarked vertices in PG)

$Candidates = \{x \in (V \models \mathcal{S}) \mid N^+(x) = \{\emptyset\}\}$
  $\forall a, b \in Candidates,\ a \neq b,\ \varphi = (a,b) \in \mathcal{P}$
    $b.offset = b.offset + |\varphi|$

  $\forall c \in Candidates$
    $TargetNode(c).RODL[RODLindex + c.offset] =$
      $OP\_EXEC$
    $RODLindex = c.offset + ExecutionTime(c)$

    $TargetNode(c).RODL[RODLindex] = OP\_SEND$

    $Receivers = \{y \in (V \models \mathcal{S}) \mid (c, y)\ in\ \mathcal{C}\}$
    $\forall r \in Receivers$
      $TargetNode(r).RODL[RODLindex] =$
        $OP\_RECEIVE$

      $RODLindex = RODLindex + \#SendBytes$

    $mark\_in\_PG(c)$
```

Fig. 2. Pseudocode representation of the scheduling algorithm

- Only one directed causal edge, in each direction, is allowed between a pair of service tasks.
- Service tasks depending on each other must not be “in phase”. Hence there may not be a phase dependency between a pair of service tasks, if we find a *path* of causal edges between the associated vertices.

Furthermore, we assume that phase dependencies are *transitive*. For instance in Figure 1 we see that service tasks “IR1” and “IR2” are in phase, as well as “IR2” and “IR3”. Due to the transitivity we conclude that “IR1” and “IR3” are also in phase. Nevertheless, we do not draw a phase edge between these vertices in order to reduce complexity of the Precedence Graph.

Figure 2 outlines in pseudocode how the schedule works. The TTP/A Scheduler runs the TTP/A Scheduling Algorithm on the Precedence Graph for several iterations. In each iteration the algorithm searches for all those service tasks that are not dependent on any other service task. We call this subset of service tasks the *Candidates*. A Candidate can be recognized by the fact, that its vertex does not possess *incoming* causal edges in the Precedence Graph. Candidates are sorted using the Earliest Deadline First algorithm.

Phase dependencies may cause a drift for the starting point of a pair of Candidates. Thus, for each pair of Candidates in phase, this one with the higher deadline is assigned the value of the *phase offset*.

Next, the Candidates are marked as done and the appropriate number of TTP/A slots for execution of each service is reserved in the RODL of the assigned target node. In case of an existing connection of any type (phase or causal) between a Candidate and some other service the scheduler inserts some TTP/A slots for the sending operation in the RODL of the Candidate's target node. In addition, the dependent service tasks, i. e., the receivers, will have to perform a receive operation in the same TTP/A slot (at the same time). Consequently, the scheduler reserves receive slots in the RODLs of the target nodes involved with the services' tasks.

The design of the TTP/A Scheduling Algorithm guarantees an implicit *collision avoidance* at the real-time communication system level. For that purpose each target node possesses a local "RODL index", which points at the last occupied TTP/A slot in the RODL.

In addition, there exists one "global index". It points at the last TTP/A Slot, which has been occupied by communication from some service at the real-time communication system. When a service execution is to be entered at the target node's RODL, the local index determines the starting slot. Then the phase offset of that service is added. The local index will be increased by the phase offset plus the number of occupied execution slots.

Moreover, the RODL index serves to determine, whether the communication system is occupied, when a target node intends to send a service's data. If the local index is greater than the global one then no other node uses the bus. The service may send immediately after the completion of its execution, the local index is increased and the global index is adjusted accordingly. Conversely – if the global index is greater – the bus is occupied by somebody else and the service has to wait to trigger its sending operation until the globally indexed TTP/A slot has passed by. Then the global index is increased by the number of used TTP/A slots and the local index is adjusted.

The execution of the Candidates leads to the release of causal dependencies. Hence in the next iteration other service tasks are defined as Candidates, which had been dependent on other services in the prior itaration. The algorithm will terminate when all service tasks are marked as done.

Finally, the tool performs a *deadline check* to verify whether the specified deadline of service tasks and dependencies hold among the calculated RODLs. Therefore, the points of start and completion, as well as the arrival times of sent data for each service and dependencies are compared to the deadline specifications in the cluster configuration description. The overall points in time must be lower than the deadlines, otherwise a deadline miss occurs. This examination will be conducted by the TTP/A Scheduler *after* the termination of the scheduling algorithm. In case of at least one deadline miss the result has to be discarded, because the algorithm could not determine a valid schedule. This indicates that the specified available resources of the environment are not sufficient for the specified application.

In short the TTP/A Scheduling Algorithm is a *greedy algorithm* with a simple, but efficient operation and data structure. It might not produce the optimal result for a schedule of the RODL. But it guarantees that all service-specific as well as dependency-specific deadlines will hold, when a valid schedule has been found.

4. CASE STUDY

In order to show the capabilities of our architecture and the scheduling and verification tool, we have devised a distributed embedded application as case study.

We use the time-triggered protocol for SAE Class A applications TTP/A [16] as communication system. TTP/A is a low-cost fieldbus for smart transducer applications that implements the interfaces using the IFS concept and provides interfaces for real-time service, configuration/planning, and diagnostics/maintenance.

4.1. Initial Set-Up

The case study hardware comprises two infrared (IR) distance sensors, a fusion service, and a display. The application shall synchronously perform a distance measurement from the two distance sensors, fuse the measurements and display the result on the display. We will use the confidence-weighted average [15] method as fusion algorithm.

The temporal requirements of the application are as follows:

1. The value shown on the display must be updated at least every 0.1s ($d_{update} = 0.1s$).
2. IR measurements must be synchronized with a precision of $\pm 1ms$.
3. The temporal accuracy d_{acc} of the value received by the display service must be 0.05s.

Figure 3 depicts the services, connections, and the mapping of services to physical nodes.

The initial set-up of the application requires four steps.

First, the firmware containing generic TTP/A communication and specific I/O services has to be programmed into the nodes. In industrial applications, this task belongs to the smart transducer vendor.

Second, all transducer nodes have to be connected to the TTP/A bus and their respective external hardware.

Third, the scheduling and verification tool is applied to the application description. In addition to the application

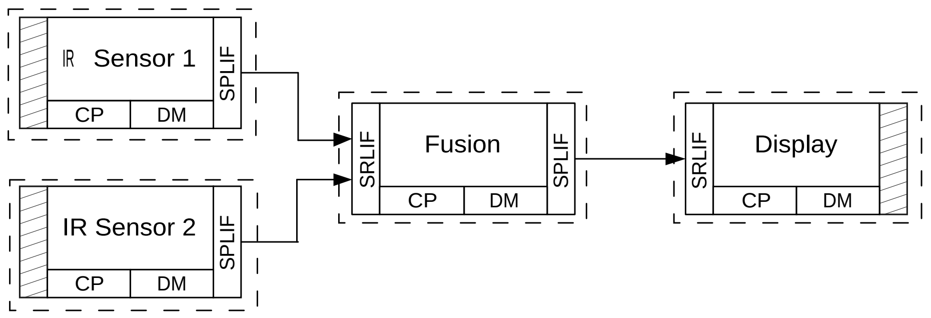

Fig. 3. Interaction of services in case study. The dashed lines mark physical nodes.

	RODL length	temporal accuracy	cycle time	
1200 Bit/s	$75,84$ ms	$\mathit{43,36}$ ms	$75,84$ ms	$\surd$
4800 Bit/s	$18,96$ ms	$10,84$ ms	$18,96$ ms	$\surd$
9600 Bit/s	$9,48$ ms	$5,42$ ms	$9,48$ ms	$\surd$

Table 1. Timing properties for created schedule with 2 IR services for different communication speeds

	RODL length	temporal accuracy	cycle time	
1200 Bit/s	$86,64$ ms	$\mathit{54,16}$ ms	$86,64$ ms	$\times$
4800 Bit/s	$21,66$ ms	$13,54$ ms	$21,66$ ms	$\surd$
9600 Bit/s	$10,83$ ms	$6,77$ ms	$10,83$ ms	$\surd$

Table 2. Timing properties for created schedule with 3 IR services for different communication speeds

descriptions, the tool also requires information on auxiliary constraints, e. g., the intended communication bandwidth, for the verification process.

Finally, if the verification holds, the schedule is uploaded to the network nodes using the configuration and planning interface of each transducer node.

Table 1 depicts the results from the Scheduling and Verification tool for the described application.

4.2. Reconfiguration with Three IR Sensors

As a case study for reconfiguration, we assume that the application should be extended to three IR sensors instead of two. Since the fusion algorithm is scalable, the application description can be easily extended by adding another IR service.

After running the Scheduling and Verification tool with the new application description we received a positive verification for communication speeds of 4800 and 9600 Bit/s (see Table 2), but not for 1200 Bit/s[2].

The updating of the previous system to the extended one requires the following steps:

First the additional node must be connected to the network. Then the plug and participate facility [17] of TTP/A assigns a free node ID to the new node. Now the modified schedules can be downloaded to the nodes, either on-line via the CP interface, if the RODL files of the IFS reside in the RAM of the smart transducer, or off-line by using conventional programming tools. Since the configuration state of the fusion service is also represented in the IFS, the configuration of the fusion service can be also changed over the CP interface, so that it knows how many source services are connected to it.

[2]We have selected rather low bit rates in this example in order to depict a case where the verification fails. The used hardware supports much higher communication speeds.

5. CONCLUSION

In this paper we proposed a model-based approach to support flexibility for a TTP/A system.

The application is described by temporal end-to-end requirements and the intended interaction of services. We have devised a formal representation of this information in XML.

The XML data is used as input to a scheduling tool that creates the static schedule and verifies the specified temporal requirements.

It is possible to change or extend the application by feeding the new description into the scheduling tool and upload the new configuration via the configuration and planning interfaces of the smart transducers in the network. This approach supports composability since the output of the scheduling tool guarantees that the timing constraints hold for the prior application and its extension.

If the available bandwidth is not sufficient to fulfill the temporal requirements of the application, the tool reports on a deadline miss, otherwise it can be guaranteed that the generated schedule meets the specified timing requirements.

Since the scheduling tool parses the XML descriptions off-line on a maintenance computer, the resource requirements of our approach are extremely low. In our case study, we used a network of five embedded AVR 8-Bit microcontrollers to demonstrate our approach.

6. REFERENCES

[1] "The Hansen report on automotive electronics," Nov. 1998, 11(9), RYE, NH USA.

[2] H. Kopetz, *Real-Time Systems, Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers, Boston, Dordrecht, London, 1997.

[3] V. Claesson, *Efficient and Reliable Communication in Distributed Embedded Systems*, Ph.D. thesis, Chalmers University of Technology, Department of Computer Engineering, Göteborg, Sweden, 2002.

[4] L. Almeida, J. A. Fonseca, and P. Fonseca, "Flexible time-triggered communication on a controller area network," in *Work-in-Progress Session of the 19th IEEE Real-Time Systems Symposium*, Steve Goddard, Ed., Madrid, Spain, Dec. 1998, pp. 23–26.

[5] Roman Obermaisser, *Event-Triggered and Time-Triggered Control Paradigms*, Springer, 2004.

[6] J. Berwanger, C. Ebner, A. Schedl, R. Belschner, S. Fluhrer, P. Lohrmann, E. Fuchs, D. Millinger, M. Sprachmann, F. Bogenberger, G. Hay, A. Krüger, M. Rausch, W. O. Budde, P. Fuhrmann, R. Mores, Bayerische Motoren Werke, DaimlerChrysler, Dependable Computer Systems, Motorola, and Philips, "FlexRay–The communication system for advanced automotive control systems," *SAE World Congress 2001, Detroit, Michigan, USA*, Mar. 2001.

[7] P. Veríssimo and A. Casimiro, "The timely computing base model and architecture," *IEEE Transactions on Computers*, vol. 51, no. 8, pp. 916–930, Aug. 2002.

[8] J. Lisner, "A flexible slotting scheme for tdma-based protocols," in *ARCS 2004 Workshop on Dependability and Fault Tolerance*, Augsburg, Germany, Mar. 2004.

[9] T. A. Henzinger, C. M. Kirsch, M. A. A. Sanvido, and W. Pree., "From control models to real-time code using Giotto," *IEEE Control Systems Magazine*, vol. 23, no. 1, pp. 50–64, 2003.

[10] H. Kopetz et al., "Specification of the TTP/A protocol," Tech. Rep., Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, Sept. 2002, Version 2.00.

[11] C. Jones, M.-O. Killijian, H. Kopetz, E. Marsden, N. Moffat, D. Powell, B. Randell, A. Romanovsky, R. Stroud, and V. Issarny, "Final version of the DSoS conceptual model," *DSoS Project (IST-1999-11585) Deliverable CSDA1*, Oct. 2002, Available as Research Report 54/2002 at http://www.vmars.tuwien.ac.at.

[12] H. Kopetz, M. Holzmann, and W. Elmenreich, "A universal smart transducer interface: TTP/A," *International Journal of Computer System Science & Engineering*, vol. 16, no. 2, pp. 71–77, Mar. 2001.

[13] W. Elmenreich, S. Pitzek, and M. Schlager, "Modeling distributed embedded applications on an interface file system," in *Proceedings of the Seventh IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, May 2004, pp. 175–182.

[14] C. Paukovits, "Modellierung und Scheduling von flexiblen, zeitgesteuerten Kommunikationsprotokollen," Bachelor's thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2004.

[15] W. Elmenreich, *Sensor Fusion in Time-Triggered Systems*, Ph.D. thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2002.

[16] H. Kopetz et al., "Specification of the TTP/A protocol," Tech. Rep., Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, Sept. 2002, Version 2.00.

[17] W. Elmenreich, W. Haidinger, P. Peti, and L. Schneider, "New node integration for master-slave fieldbus networks," in *Proceedings of the 20th IASTED International Conference on Applied Informatics (AI 2002)*, Feb. 2002, pp. 173–178.