Monitoring and Configuration in a Smart Transducer Network

Roman Obermaisser, *Member, IEEE*, Philipp Peti, Wilfried Elmenreich, Thomas Losert

Abstract – **There is a strong necessity for monitoring, runtime configuration, and maintenance of smart transducer networks. This paper describes tools for establishing these tasks in distributed real-time systems. A time-triggered architecture is presented, that accomplishes deterministic, reproducible monitoring of a smart transducer network without probe effects. This architecture allows access to the relevant node internal information to render all conditions observable, which cause a specific system behavior. Timeliness during dynamic reconfiguration and maintenance is preserved, which allows evolutionary changes and maintenance of the system during real-time operation. The architecture comprises both a local application and a CORBA client/server communication. We present a case study providing insight into a concrete implementation of this architecture based on the fieldbus protocol TTP/A.**

Index Terms—Smart Sensor, Runtime Configuration, Time-Triggered Architecture, Interface Design, Monitoring, CORBA, Distributed Real-Time Systems

1 Introduction

In 1982 Wen H. Ko and Clifford D. Fung introduced the term "intelligent transducer" [1]. An intelligent or *smart* transducer is the integration of an analog or digital sensor or actuator element and a local microcontroller that contains the interface circuitry, a processor, memory, and a network controller in a single unit. The smart sensor transforms the raw sensor signal to a standardized digital representation, checks and calibrates the signal, and transmits this digital signal via a standardized communication protocol to its users [2].

Smart transducer technology implicates the development of transducer networks, that allow monitoring, plug-and-play configuration and the communication of digitized transducer data. Such a smart transducer network provides a framework that helps to reduce the complexity and cost of large distributed real-time systems.

Monitoring and debugging of distributed embedded real-time systems differ significantly from debugging and testing programs for desktop computers, because only few interfaces to the outside world are present [3]. In addition, a distributed system contains multiple locations of control and therefore conventional breakpoint techniques result in an undesired modification of the timing behavior. This indeterministic effect is called the "Probe Effect" [4], [5] or the "Heisenberg Uncertainty" [6] applied to software. Therefore a vital property of a convenient monitoring environment is the absence of disturbances or intrusions on the system behavior. Users expect tools to avoid the probe effect and to incorporate a deterministic and reproducible behavior.

Much research has focused on monitoring, debugging, configuration, and maintenance of distributed real-time systems. However, existing tool sets generally neglect a subset of the major monitoring qualities like reproducibility, determinism and the elimination of the probe effect. The tool sets presented in Section 2.4 restrict observations to the bus contents, although determinism involves the ability to observe all conditions, which cause a specific system behavior. These solutions eliminate the probe effect by renouncing deterministic monitoring as defined in [7]. They also lack mechanisms for preserving timeliness during dynamic reconfiguration and maintenance. This fact makes these approaches unsuited for online reconfiguration and maintenance in a hard real-time environment, i. e. modifications to the system require a system shut down.

We have developed a tool set for hard real-time environments, which integrates the above mentioned functions while exploiting the properties of the time-triggered protocol TTP/A to establish the major monitoring qualities. In contrast to existing tools relevant node internal information is accessible while preventing any modification to the behavior in either the temporal or value domain. On-the-fly configuration and maintenance can be performed while preserving the timeliness in the exchange of real-time data. Therefore, evolutionary changes of the system can occur during real-time operation.

This paper describes tools and architectures to accomplish this task and is structured as follows. In Section 2 related work in the field of monitoring of distributed real-time systems is presented. Section 3 gives an overview of smart transducer interfaces. Section 4 describes the Time-Triggered Architecture with Internet tool support. In Section 5 the implementation of monitoring tools as a case study is described. Section 6 gives an outlook to future work and Section 7 concludes the paper.

2 An Overview on Monitoring

Monitoring is the process of gathering information about a program's execution that cannot be obtained by static analysis of the source code.

All four authors are with the Real-Time Systems Group of the Insitut für Technische Informatik at the Vienna University of Technology.

2.1 Purposes of Monitoring

One function of monitoring is the collection of run-time information of a real-time system for testing and debugging. Monitoring is often an effective procedure for locating incorrect system behavior and serves the purpose of a debugging tool [5]. In a critical real-time system it can provide additional confidence for the validity of a static analysis. In non-critical real-time system software dynamic debugging and testing may be preferred due to lower cost and complexity.

Distributed real-time systems require the possibility for on-the-fly configuration and maintenance without a system shut down [8]. This allows the accommodation of evolutionary changes of the system, which is especially important for networks with a long expected lifetime. Configuration is aided by electronic data-sheets that contain all relevant information of a transducer [9]. This information is located on the chip in persistent memory and may not be separated from the chip. Hence human errors associated with entering sensor parameters manually are completely eliminated. Losing transducer paper data sheets is no longer a concern. Enhanced capabilities like plug and play are possible [10].

Monitoring also aids in the automatic detection and handling of anomalous system states. If erroneous conditions described by monitoring constraints occur, a monitoring application can initiate corrective actions.

2.2 Monitoring Requirements

Useful monitoring tools must satisfy certain qualities [11], [7].

Elimination of the Probe Effect: The act of observing a distributed real-time system can change its behavior. Existing errors can be hidden and new errors can be introduced. Due to multiple locations of control, the halting of a single control path introduces a temporal distortion. Therefore a vital property of a convenient monitoring environment is the absence of disturbances or intrusions on the system behavior.
Deterministic Monitoring: Deterministic monitoring means that all conditions are observed, which cause a specific system behavior. For deterministic monitoring all entities must be observed with respect to contents, order and timing.
Reproducible Monitoring: Reproducible monitoring is advantageous for debugging, since erroneous conditions can be deterministically reproduced. For sustaining reproducibility in a conventional sequential program, it is sufficient to start at the same initial state and to provide the same inputs. In a real-time system, there is the additional requirement for a reproducible timing behavior. Therefore, the inputs have to be reproduced with respect to contents, order and timing. Tasks have to be executed in the same order and the probe effect must be eliminated.

2.3 Monitoring Approaches

Various monitoring mechanisms are available and categorized into three types: hardware, software, and hybrid monitors. These monitoring mechanisms can be distinguished in terms of cost, reusability and the elimination of the probe effect.
Hardware Monitoring: Separate hardware objects observe the distributed system. Tsai et al. [12] use physical probes that are connected to the processors and memory ports. The main advantage of this approach is its non-intrusive nature. Since information is read from busses and processor ports only, processor internal data is invisible to hardware monitoring.
Software Monitoring: Resources are shared with the monitored system. Software probes are inserted into the code to gather information. The main disadvantage of this approach is the problem of predicting the perturbations caused by the additional code. However this approach offers flexibility, portability, lower cost, and easier design than hardware monitors. A tradeoff between predictability and cost exists in the final system. If the code pieces for software monitoring are removed from the target system after testing, the probe effect may occur. On the other hand remaining of monitoring code results in increased usage of hardware resources. An example for software monitoring is presented in [13].
Hybrid Monitoring Hybrid monitors are designed to overcome the shortcomings of the above mentioned approaches. Typically hybrid systems consist of an independent hardware device that receives monitoring information generated by software probes. Software probes are inserted into the monitored software objects [12]. So called "Triggers", which are implemented in software, assist the dedicated hardware on recording important events. The code used for monitoring must also be resource adequate in order to avoid the probe effect. This way the level of abstraction is increased and the amount of the recorded information is decreased [3]. Functional limitations of hardware monitoring are eliminated and the internal state of the target system and intermediate results of computations are accessible. Nevertheless, disadvantages of increased cost and usability for a specific target system only remain.

2.4 Related Work

A network configuration and monitoring tool suite developed for the fieldbus protocol LIN is described in [14], [15]. A hardware monitoring tool records the contents and the points of time of occurrence of LIN frames. This bus traffic is stored in log-files for later analysis. The internal state of nodes remains hidden. For configuration purposes a tool chain allows the generation of a target image out of a specification of communication requirements. This target image is loaded into the nodes during service mode. In contrast to the monitoring solution described in this paper no online maintenance and reconfiguration is supported.

For CAN bus systems the CANalyzer tool [16] is available. It allows observing and analyzing of bus traffic. During passive mode no probe effect can occur. The internal state of the nodes remains hidden, if it is not transmitted on the network. Determinism and reproducibility are limited, since conditions contributing to a specific system behavior are not necessarily broadcast on the bus.

A hardware monitoring tool developed for the time-triggered fieldbus TTP/A is described in [17], [18]. It records and stores all data on the real-time bus for later analysis. Trigger conditions are employed for filtering purposes, i. e. to reduce the amount of recorded data. Due to the fact that only information transmitted on the bus is accessible, node internal data values cannot be inspected.

Table 1 summarizes the capabilities of the described tool sets and compares them to the properties of the monitoring solution presented in this paper.

Tool	Bus Contents Observ.	Relevant Internals Observ.	Dynamic Config. Support
CANalyzer	yes	no	no
LINspector	yes	no	no
TTP/A HW	yes	no	no
TTP/A Hybrid	yes	yes	yes

Table 1: Comparison of Monitoring and Configuration Tools

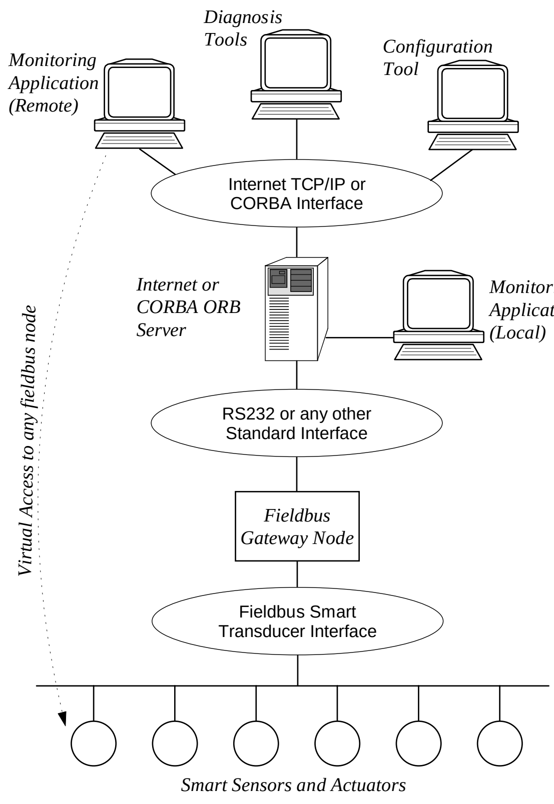

Diagnosis Tools
Monitoring Application (Remote)
Configuration Tool
Internet TCP/IP or CORBA Interface
Internet or CORBA ORB Server
Monitoring Application (Local)
RS232 or any other Standard Interface
Virtual Access to any fieldbus node
Fieldbus Gateway Node
Fieldbus Smart Transducer Interface
Smart Sensors and Actuators

Figure 1: System Architecture

3 System Architecture with Internet Tool Support

Figure 1 depicts the system architecture of our approach. We identified 4 levels above each other, with a well-defined interface between each two levels.

The first level is the fieldbus level. It comprises transducers (sensors and actuators) which are interconnected by a digital fieldbus. The transducers consist of a physical sensor or actuator integrated with a microcontroller unit and a communication interface. The microcontroller contains the "intelligence" to transform the raw sensor signal to a standardized digital representation respectively transform a digital control value to an appropriate actuator signal.

All smart transducers are accessible via a standardized smart transducer interface. In order to support complexity management and composability, it is useful to specify distinct interfaces for functional different services [19]. Three types of interfaces can be distinguished: The *Real-time Service* interface (RS) provides the timely real-time services to the component during the operation of the system. The *Diagnostic and Maintenance* interface (DM) opens a communication channel to the internals of a component. It is used to set parameters and to retrieve information about the internals of a component, e. g., for the purpose of fault diagnosis. This interface is available during system operation without disturbing the real-time service. The *Configuration and Planning* interface (CP) is necessary to access configuration properties of a node. It is used for initial configuration and for system reconfiguration.

The second level is the gateway level. The fieldbus gateway node is connected to the fieldbus and contains the software and hardware to establish a connection via a standard PC interface. We used an RS232 serial connection. Other possible connections are RS485, USB, IEEE 1394, or a wireless connection with IrDA or radio signals. The used protocol for this connection shall guarantee real-time behavior.

The next level hosts a PC running a communication module and local tools for monitoring and configuration and a server module. Because the timing behavior of the communication from the local tools into the fieldbus network provides a deterministic timing behavior, the local tools can support deterministic real-time monitoring. However, compared to access at fieldbus level data is accessible at a lower bandwidth.

The communication module running on the PC will be a CORBA (Common Object Request Broker Architecture) ORB server. The OMG (Object Management Group), the largest standardization group for CORBA services, currently examines a standard for smart transducer interfaces [20]. Our interface will conform to this future standard.

The CORBA system enables the transparent Internet access of remote service applications (level four) to the fieldbus network. This interface is independent of the employed protocols and physical layers at fieldbus level.

4 Case Study

The goal was to develop a set of tools allowing rapid development, monitoring and dynamic reconfiguration of TTP/A networks. This task was accomplished by devising a hybrid monitoring solution combining monitoring hardware with instrumentation code. The tools were designed according to the presented system architecture with Internet tool support. Emphasis was set on the realization of the monitoring qualities described in section 2.2, namely reproducibility, determinism and the elimination of the probe effect. The tools were implemented as part of the development of smart TTP/A transducer networks. These networks controlled an autonomous mobile robot and a demonstrator with an arm prosthesis [17], [18], [21].

4.1 Fieldbus Level - TTP/A Bus

TTP/A is a time-triggered master/slave communication protocol for fieldbus applications that uses a time division multiple access (TDMA) bus arbitration scheme [22]. It is possible to address up to 255 nodes on a bus. One single node is the active master. This master provides the time base for the slave nodes. The communication is organized into rounds. Bus access conflicts are avoided by a strict TDMA schedule for each round. A round consists of several slots. A slot is a unit for transmission of one byte of data. Data bytes are transmitted in a standard UART format. Each communication round is started by the master with a so-called fireworks byte. The fireworks byte defines the type of round.

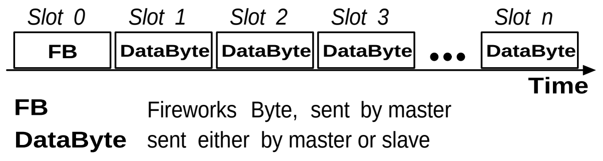

Slot 0
Slot 1
Slot 2
Slot 3
Slot n
FB
DataByte
DataByte
DataByte
DataByte
Time
FB Fireworks Byte, sent by master
DataByte sent either by master or slave

Figure 2: A TTP/A Round

A TTP/A round (see Figure 2) consists of a configuration dependent number of slots and an assigned sender node for each slot. The configuration of a round is defined in the RODL (ROund Descriptor List). The RODL defines which node transmits in a certain slot, the semantics of each individual slot, and the receiving nodes of a slot. RODLs must be configured in the slave nodes prior to the execution of the corresponding multipartner round.

A master/slave round is a special round with a fixed layout that establishes a connection between the master and a particular slave for reading/writing monitoring or configuration data, e. g. the RODL information. In a master/slave round the master addresses a data record in the IFS format and specifies an action like reading, writing or executing on that record.

The master/slave rounds establish the CP and DM interface to the transducer nodes. Master/slave rounds are intended to be scheduled periodically between multipartner rounds as depicted in Figure 3.

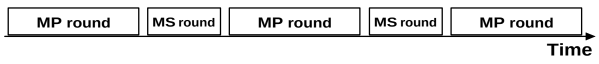


Figure 3: Recommended TTP/A Schedule

The TTP/A protocol [22] offers a unique addressing scheme for all relevant data of a node like communication schedules, calibration data, and I/O properties. This addressing scheme is called Interface File System (IFS) [2]. The IFS provides a universal interface to the TTP/A network for configuration and maintenance tools as well as for applications running local on a node. The IFS is structured in a record-oriented format. The smallest addressable unit is a record of 4 bytes. All nodes contain several files with a number of records that can contain information for automatic configuration, similar to the idea of Transducer Electronic Data Sheets in the IEEE 1451.2 standard [23].

4.2 Gateway Level - TTP/A Gateway Component

The TTP/A gateway component represents the interface between the workstation level and the fieldbus level. It performs the task of the fieldbus gateway node described in section 3. The gateway node is designed to support the TTP/A master by shifting load from it. Requests for accessing the IFS are queued at the gateway component and forwarded to the master at points in time determined by the master. Since the TTP/A master controls the rate at which information is exchanged with the gateway component, no violation of timing constraints with respect to the TTP/A communication can occur. Probe effects are avoided, because timely execution of the protocol code in the master can be ensured despite monitoring activities. The gateway component also performs the prioritization of the requests for IFS accesses according to the different types of interfaces introduced in Section 3. Time critical operations of the RS interface are forwarded prior to the operations of the DM and CP interfaces.

As a consequence of the preprocessing and queuing performed by the gateway component the additional instructions needed for monitoring take less than 15 μs in the master implementation on the Microchip PIC16F84 microcontroller unit (clocked at 8 MHz). These are 2.2% of the processing time of the microcontroller unit during a time slot of 677 μs (communication at 19200 Baud). The cost for the additional hardware is minimized, since commercial off the shelf components (COTS) are utilized. An Atmel AT90S2313 microcontroller unit was used as gateway component.

4.3 Workstation Level

The workstation level consists of both a local monitoring application and a CORBA server module representing the interface to remote monitoring clients. The local monitoring application allows real-time monitoring, maintenance and dynamic configuration without indeterministic delays caused by the Internet. Remote client applications performing monitoring, diagnosis or configuration tasks access the Interface File System of the fieldbus level by sending requests to the CORBA Server Module (see Figure 4).

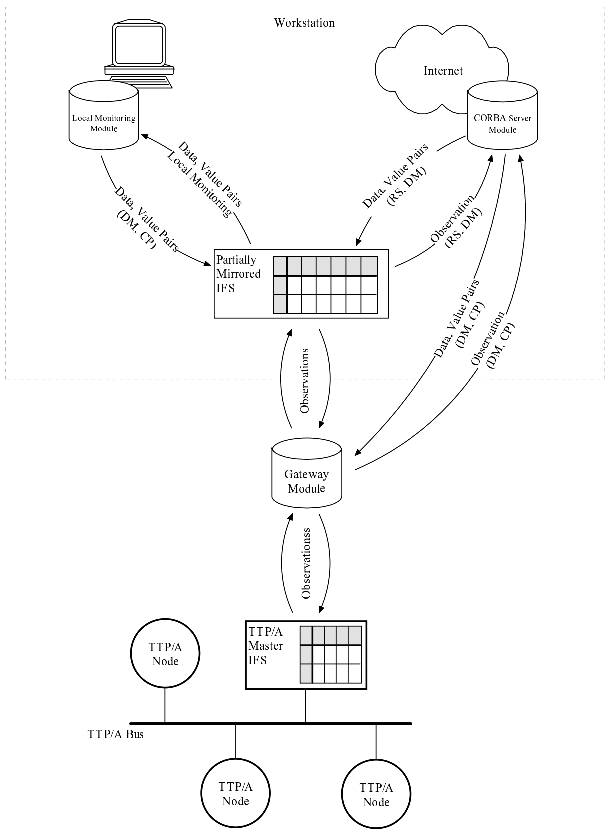


Figure 4: Workstation Interacting with other Levels

Records accessed through the Real-Time Service Interface (RS) are placed in a shared memory called the "Partially Mirrored IFS". Data elements stored in this data structure are updated at a constant bandwidth. Thereby the age of the data elements is bounded by the update period. The maximum age is independent of the user's access rate.

The contents of the "Partially Mirrored IFS" are observations. An observation is any property of a relevant state variable that is observed by a smart transducer (ST). An observation records the state of a state variable at a particular instant, the point of observation. An observation can be expressed by the atomic triple <name of the observed state variable, observed value, time of observation>. An observation is stored in a record of the IFS within an ST and is normally periodically updated by internal encapsulated processes of the ST. The observed value is contained in the record and the time of observation is the time of updating the record by the internal process of the ST.

CORBA Server Module: The CORBA Server establishes a connection to the remote clients. Client application automatically benefit of the properties of the CORBA concept. Among the advantages are scalability, maintenability, fault tolerance, independence of the physical layer, language independence and transparency of the network layer [24].

In our architecture the maximum age of data elements gained at the fieldbus level is known up to the CORBA server. In conjunction with real-time CORBA it is possible to additionally achieve end-to-end predictability, i. e. the temporal behaviour of all transmission activities are deterministic. This results in knowledge about the maximum data element age at the clients.

Due to the nature of the Internet, the timing of the master/slave access is indeterministic at this point. The timing unpredictability is caused by the bursty network traffic of the Internet. This lack of temporal certainty is non-critical because access to the configuration interface of the cluster nodes is not a time critical task (see Section 3).

Local Monitoring Module: The local monitoring module is superior to remote monitoring applications in terms of temporal predictability. The values of the interface file system are presented to the user in a graphical representation. The tool also offers configuration capabilities. The local monitoring application serves the purpose of visualization of the real-time data. This visualization takes place with a scalable window of the information space. This approach to adjust the proper level of detail is called "Pan and Zoom approach" [25]. Operators are aided in the comprehension of events on the network. The monitoring functions of testing, debugging, configuration, maintenance and electronic datasheet extraction are enabled. Configuration and maintenance are achieved by establishing the possibility to write directly into the interface file system of a master node.

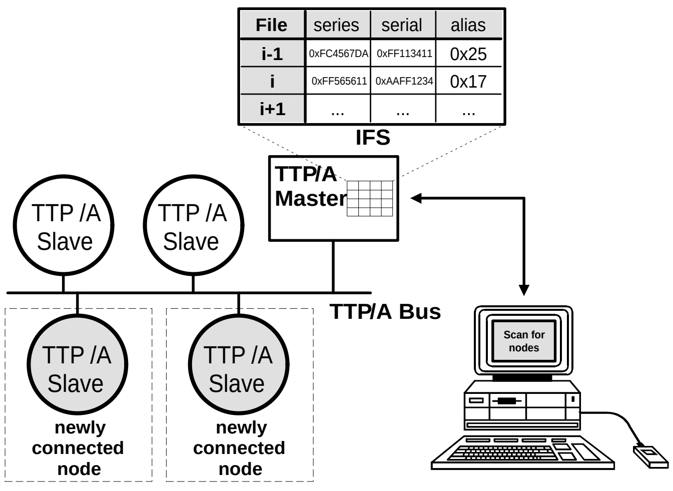
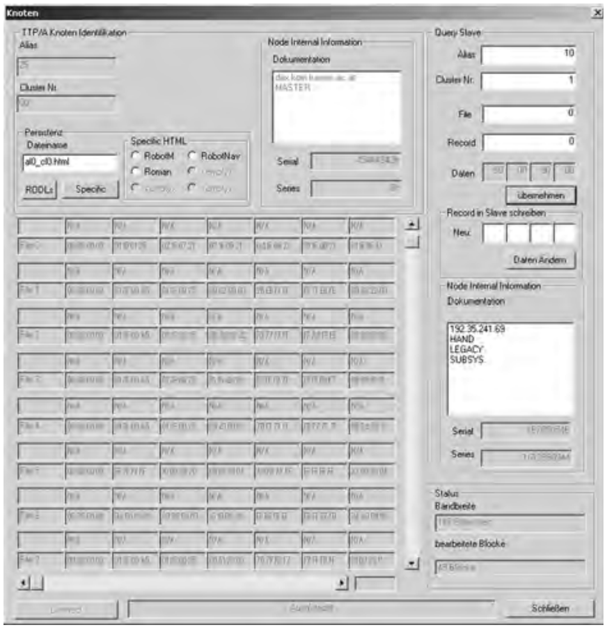

File	series	serial	alias
i-1	0xFC4567DA	0xFF113411	0x25
i	0xFF565611	0xAAFF1234	0x17
i+1	...	...	...

Figure 5: Dynamic Configuration Example: Plug'n'Play

Figure 5 illustrates the concept of using the IFS for dynamic configuration purposes. The user can initiate a scan for new nodes followed by a configuration of newly connected nodes according to their transducer electronic datasheets. This approach is based upon the work presented in [26]. As a consequence of the user command the master scans for recently added nodes and automatically assigns a unique node identifier (alias) to each found node without affecting the real-time services. In addition to the series and serial number (assigned by the hardware manufacturer) the alias of the newly connected node is stored in a record of the IFS providing feedback for the user. As soon as the alias is assigned the node can be configured according to its electronic datasheet using master/slave accesses. The alias is necessary for addressing a particular node during a master/slave access (see Section 4.1).


Knoten

TTP/A Knoten Identifikation
Alias: 25
Cluster Nr: 00

Persistenz
Dateiname: al0_cl0.html

Node Internal Information
Dokumentation:
MASTER

Query Slave
Alias: 10
Cluster Nr.: 1
File: 0
Record: 0

Node Internal Information
Dokumentation:
192.35.241.69
HAND
LEGACY
SUBSYS

Figure 6: User Interface of the Monitoring Application

Figure 6 shows the user interface of the local monitoring application. Most of the window space is consumed by the visualization of the status information (IFS). The electronic datasheet of the master node and details of the monitoring process are also displayed. Both electronic datasheets and file system entries of other nodes can be queried.

4.4 Establishment of the Monitoring Requirements

The *probe effect* was eliminated by executing the monitoring code in TTP/A nodes only during idle times. As a consequence no modification of the temporal behavior could occur. The determination of the necessary resources for allowing both time critical real-time operations and monitoring during peak load scenarios was possible by the a-priori knowledge of the processing time required for the execution of the monitoring code.

Determinism was established by mapping all relevant data elements into the interface file system (see Section 4.1). This eases the monitoring task significantly. The IFS provides a fully specified interface in both the value and time domain between the application and the protocol code of the node. Reading this IFS periodically enables the system developer to gain insight of the node's behavior. By the provision of a global time in all TTP/A nodes events can be time stamped. Therefore all conditions causing a certain system behavior are observable with respect to their values and the point in time of monitoring. In contrast to the monitoring solutions presented in Section 2.4 the internal state of nodes need not be broadcast on the bus to be observable.

Reproducibility was ensured by the properties of the TTP/A protocol. The time-triggered architecture results in a predefined execution order of tasks and no need for explicit synchronizing constructs (e. g. semaphores).

5 Outlook

Because the Internet interface opens a wide gap for security threats like data interception and manipulation, the connection to the cluster interface will be protected by strong encryption. The performance of the local server respective the client workstation will suffice for the encryption algorithms. On the other hand the access to the cluster should be very comfortable. To support this we plan to implement an authentification service with different access levels.

As a further improvement the server module may be implemented on a single board computer running the CORBA-Object Request Broker on embedded Linux – thus reducing cost by replacing the workstation.

6 Conclusion

It can be stated that monitoring, configuration and maintenance tools are essential development components. We have devised a tool set for hard real-time environments, which integrates monitoring, maintenance and configuration functions while exploiting the properties of the time-triggered protocol TTP/A to establish the major monitoring qualities. In contrary to existing tools relevant node internal information can be accessed while preventing any modification to the behavior in either the temporal or value domain. Configuration and maintenance of the running system can be performed, thus evolutionary changes of the system can occur during real-time operation. The time-triggered architecture is well suited for the implementation of monitoring, online reconfiguration and maintenance because protocol mechanisms (e. g. master/slave access) ensure hard real-time constraints despite monitoring. To gain reproduciblity explicit synchronization is avoided. In TTP/A implicit synchronization is established by a static temporal control structure ensuring mutual exclusion and precedence requirements without any further need for explicit synchronization.

A container for the significant part of the information space is formed by the IFS. The IFS is the key datastructure for our TTP/A tool implementation, because it makes the node data for monitoring, configuration and maintenance accessible in a uniform manner. In order to provide remote CORBA clients with information about the age of an IFS record, the time of the corresponding observation is transmitted in addition to the record data. Therefore, the age of a record is known at remote clients despite indeterministic delays caused by the internet.

7 Acknowledgments

We would like to give special thanks to our colleague Michael Paulitsch for constructive comments on earlier versions of this paper. This work was supported in part by the Austrian Ministry of Science, project TTSB and by the European IST project DSoS under contract No IST-1999-11585.

8 References

[1] W. H. Ko and C. D. Fung. VLSI and intelligent transducers. *Sensors and Actuators*, (2):239–250, 1982.

[2] H. Kopetz, M. Holzmann, and W. Elmenreich. A universal smart transducer interface: TTP/A. *International Journal of Computer System Science & Engineering*, March 2000.

[3] H. Thane. *Monitoring, Testing and Debugging of Distributed Real-Time Systems.* Phd thesis, Mechatronics Laboratory, Royal Institute of Technology, Stockholm, Sweden, May 2000.

[4] J. Gait. A probe effect in concurrent programs. *Software Practice and Experience*, 16(3):225–233, March 1986.

[5] C. E. McDowell and D. P. Helmbold. Debugging concurrent programs. *ACM Computing Surveys*, 21(4):593–622, December 1989.

[6] C. H. Ledoux and D. Stott Parker. Saving traces for Ada debugging. In *Ada in Use (1985 International Ada Conference)*, pages 97–108, Cambridge, England, May 1985. Cambridge University Press.

[7] M. Mansouri-Samani and M. Sloman. Monitoring distributed systems. *IEEE Network*, 7(6):20–30, November 1993.

[8] J. Kramer and J. Mageee. Dynamic configuration for distributed systems. *IEEE Transactions on Software Engineering*, SE-11(4):424–436, 1985.

[9] K. Lee. IEEE 1451: A standard in support of smart transducer networking. *Proceedings of the 17th IEEE*, 2:525–528, 2000.

[10] R. Johnson. Building plug-and-play networked smart transducers. *Sensors Magazine*, October 1997.

[11] C. Glawan. Monitoring von Echtzeitbetriebssystemen. Master's thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2000.

[12] J. P. Tsai, K. Fang, and H. Chen. A noninvasive architecture to monitor real-time distributed systems. *Computer*, 23(3):11–23, March 1990.

[13] H. Beier and T. Bemmerl. Software monitoring of parallel programs. In *Proceedings of CONPAR '88*, pages 71–78, Manchester, England, 1988.

[14] A. Rajnak J. Specks. LIN protocol development tools and software interfaces for local interconnection networks in vehicles. 9th International Conference on Electronic Systems for Vehicles, Baden-Baden, 2000.

[15] Audi AG, BMW AG, DaimlerChrysler AG, Motorola Inc., Volcano Communication Technologies AB, Volkswagen AG, and Volvo Car Corporation. LIN specification and LIN press announcement. SAE World Congress Detroit, http://www.lin-subbus.org, 1999.

[16] IXXAT, Inc. CANalyzer – new product descriptions. The Embedded Systems Conference San Francisco, 2001.

[17] R. Obermaisser. Design and Implementation of a Distributed Smart Transducer Network. Master's thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2001.

[18] P. Peti. Monitoring and Configuration of a TTP/A Cluster in an Autonomous Mobile Robot. Master's thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2001.

[19] H. Kopetz. Software engineering for real-time: A roadmap. *IEEE Software Engineering Conference, Limmerick, Ireland*, 2000.

[20] Objective Interface Systems, TTTech Computertechnik, and VERTEL Corporation. Smart transducers interface. *OMG TC Document orbos/2001-06-03*, July 2001. Supported by Technische Universität Wien. Available at http://www.omg.org.

[21] L. Schneider. Real time Robot Navigation with a Smart Transducer Network. Master's thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2001.

[22] H. Kopetz et al. Specification of the TTP/A protocol. Technical report, Technische Universität Wien, Institut für Technische Informatik, March 2000. Available at http://www.ttpforum.org.

[23] L. H. Eccles. A brief description of IEEE P1451.2. *Sensors Expo*, May 1998.

[24] J. Siegel. *CORBA 3: Fundamentals and Programming.* John Wiley and Sons, Heidelberg, 1999.

[25] L. Bartram, A. Ho, J. Dill, and F. Henigman. The continuous zoom: A constrained fisheye technique for viewing and navigating large information spaces. In *ACM Symposium on User Interface Software and Technology*, pages 207–215, 1995.

[26] W. Elmenreich, W. Haidinger, P. Peti, and L. Schneider. New node integration for TTP/A networks. Technical Report 5, Technische Universität Wien, Institut für Technische Informatik, 2001.