Real-Time Communication with Direct Publish/Subscribe Event Service

Marc Schanne*
schanne@fzi.de

17. October 2005

Abstract

Currently real-time communication in embedded systems is gaining in importance. With the improvement of object-oriented programming for real-time and safety-critical system environments a distributed, component oriented design allows more dependable system engineering. In addition to schedulability analysis and thread cooperation in local controllers remote communication becomes an important aspect.

This position paper proposes a pattern oriented design and descriptive development for an event-interaction based on a decentralized, asynchronous publish/subscribe programming model with a logical event channel. With the introduction of a direct publish/subscribe event service in networked real-time embedded systems the benefits of this communication scheme are utilized for a wide range of safety critical high-integrity systems in automotive, avionics, telecommunications, and process control.

1 Introduction

Distributed high-integrity systems pervade daily life and are gaining in importance. Failure can cause loss of life, environmental harm, or significant financial loss. Research and analysis work done in recent years forecast trends in embedded systems roadmaps [5, 27] and strategies for European research in this area [10]. The expected increasing degree of heterogeneity and networking in embedded systems demand a sound networking model to define communication in high-integrity systems. The use of networked sensors and actors require distribution and reliable and timely communication.

The publish/subscribe event service described in this paper is part of an ongoing research project partially funded by the European Commission in the 6th framework program. With High Integrity Java Applications[1] (HIJA) [7, 18] the project partners plan to increase the reliance of future networked real-time embedded systems. A requirments review [22] discusses standards and possibilities for communication, synchronization, real-time, networks, and fieldbuses in embedded systems and motivates the asynchronous publish/subscribe event service. The possible improvements are apparent: 1) smart sensors and actors allow processing closer to data source or sink, 2) with error handling in the components it is possible to increase dependability of the complete system, 3) structuring from distributed components allows more complex systems, 4) scalability can be improved by easy addition of components with replicated functionality, and 5) modular design simplifies maintainability with component replacement.

To assist these technical advancements in the software development process, this paper proposes a descriptive solution for hard real-time systems. The real-time event service [23] introduced in section 3 allows the engineering of distributed applications on top of conventionally used COTS hardware and software components in embedded systems of automation, automotive and avionics. Based on a collection of requirements on networked systems in these areas (see next section), a concept for the description of a distributed high-integrity application

*Software Engineering, FZI Forschungszentrum Informatik, Haid-und-Neu-Straße 10-14, 76131 Karlsruhe, Germany

[1] IST-511718

under hard real-time constraints is provided in section 4. With additional static scheduling analysis, it is possible to guarantee dependable systems.

2 Requirements and Related Work

The essential requirement for distributed systems is the availability of inter-process communication [26]. Concurrent and networked services enable systems with higher performance, dependability, scalability and cost efficiency [24]. This section introduces general conditions in the area of networking and real-time for high-integrity systems. These requirements are used to evaluate related work and motivate the event service design in the next section 3.

Real-time environments require a real-time network system with end-to-end timing guarantees. The network model developed in HIJA provides a synchronous communication with real-time remote method invocation (RT-RMI), and this paper introduces another connectionless mode for communication. With an asynchronous event notification mechanism, it is possible to utilize parallel, non-blocking work in the distributed system and allow a direct event communication following the publish/subscribe communication pattern. With low protocol overhead, this communication profits from the often broadcast oriented communication networks and fieldbuses in embedded systems and provides network real-time guarantees to the application layer.

2.1 Networksystems and Fieldbuses

The network is the basis for a distributed system. It supports communication between nodes and can be seen as a critical resource because the loss of communication results in the loss of global system services [9]. In addition to physical connections, a network consists of local connection interfaces, so that nodes can use a layered protocol providing a dependable, safe, efficient, and (for real-time networks) timely communication service.

An example of a layered protocol is the ISO/OSI reference model of communication, built with 7 layers from physical to application [21]. In the area of embedded systems, a reduced protocol stack with three layers and less protocol overhead is common. Several network systems and fieldbuses are used and the following selection is organized by application domains and is not exhaustive.

Avionic ARINC629 [1] is a databus standard developed by Boeing for simple cabling and control in the Boeing 777 aeroplanes.

Automotive Different new developed protocols, such as, the **Time-Triggered Protocol** (TTP/C[2]) [29] or the mixed event and time triggered **FlexRay** [2] protocol compete with established protocols like the **Controller Area Network** (CAN) [20] or the **Time-Triggered CAN** (TT-CAN).

Automation With, for example, **ProfiBus**, and other specialized fieldbuses [28] for automation and control there are many different systems available. In addition simple cabling is a new trend towards standardization based on Internet technology (**Ethernet/IP**) is becoming more commonplace.

Next generation network systems, e.g. **AFDX** (ARINC664) for deterministic Ethernet [4], and other one-to-one connection networks are not in the focus of the proposed real-time event service. The additional effort in communication infrastructure allows more complex protocols, such as, real-time RMI [30]. It's possible to use the event service but it's not required.

2.2 Related Work

In the area of platform-independent communication related work can be found in the Object Management Group (OMG) and their definition of an object/service information model for heterogeneous applications in different programming languages and on variable platforms. The real-time extension of the Common Object Request Broker Architecture (RT-CORBA) [13, 17] addresses real-time and safety-critical systems.

Primarily, CORBA supports portable development of distributed application with synchronous client/server interaction. With integration of the CORBA Messaging extension [11] in the current

[2] C characterize the SAE safety level

CORBA specification version 3 [14], the OMG tries to cope with scalability and large-scale distributed systems. With callback and polling this specification adds two asynchronous request models to the preserved synchronous request invocation models. With additional CORBA services, the OMG defines now an interface for event communication with an event service and notification service [15, 16]. These services build onto synchronous point-to-point communication and are part of the Object Request Broker (ORB) functionality. It requires a complete CORBA infrastructure and as a result of the TCP/IP orientation of CORBA and the complex concepts for platform-independence this approach has to be critically analyzed for use in embedded systems. One attempt of the OMG to deal with this problem is the MinimumCORBA specification [12], but main objectives differs from RT-CORBA, and the use of real-time in systems with minimal resources is not clear. Low resources and often used broadcast communication networks and fieldbuses need additional expenses for use with the Internet Inter-ORB protocol (IIOP) for one-to-one connections.

Hardware related solutions, for example, the Open System Architecture - Platform for Universal Services (OSA+) [3], provide a service oriented micro kernel architecture. Based on a micro kernel API with six fundamental functions, an object-oriented interface for use in distributed high-integrity systems has to be implemented first.

3 Direct Publish/Subscribe

We propose a direct publish/subscribe event service to meet the requirements of real-time and distributed high-integrity embedded systems. It is designed without any central administration unit or persistence mechanism. Unlike the RT-CORBA [17] approach, which is based on a synchronous communication mechanism, the real-time event service benefits from underlying many-to-many networks. It defines logical event channels to group the interaction with events by topic [6]. The event channel specifies parameters for events and decentralized maintenance units. In each communication node, active objects are responsible for dispatching the received events and handling in real-time. Figure 1 introduces the event channel object with

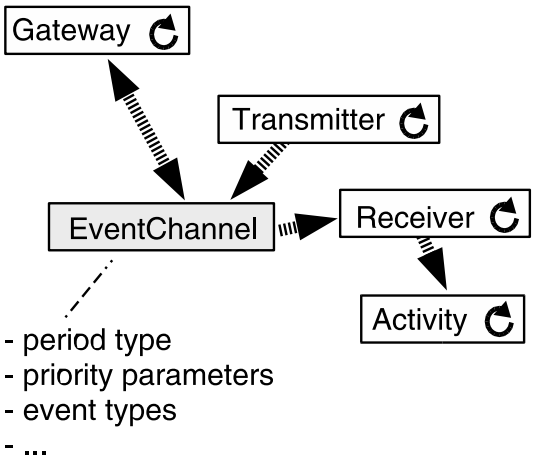

Figure 1: Event channel, parameters.

associated parameters. The event service supports periodic and sporadic[3] event transmission models and maps these on the schedulable objects in the real-time Java VM.

3.1 Design

[23] presents a component-view of the software pattern defined for the event service, and this section focuses on the active objects as well as the queueing connectors introduced for receiving and handling events in timely manner. The communication network interface (CNI) is the basic component that provides direct broadcast and bytearray-oriented network access, a BroadcastSocket encapsulates the access to the physical network system or fieldbus on the Java side. This interface is used by the proposed receiver collective to dispatch events out of the low-level input-buffer in the CNI and forward the unmarshaled event objects to a priority-sorted FIFO queueing system. Figure 2 depicts the structure and dataflow and introduces the subsequent activity manager thread, which controls a pool of activities (i.e., threads with different priorities) to handle events according to their priorities[4].

As suggested in [19], for a clear design and overall predictability, a segmentation of the real-time application in initialization phase and mission phase is

[3] with a minimum inter-arrival time
[4] parameters of their event channel

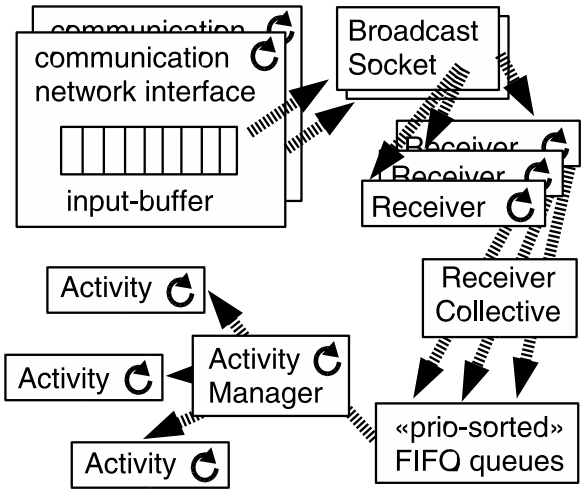

Figure 2: Active objects and data flow.

required. The initialization phase is used for non-time-critical activities, such as, creation and configuration of all mission real-time threads, memory objects, and event handlers. It is assumed that no mode changes in the application follow, and the application can be scheduled statically. To guarantee hard real-time constraints the definition and configuration of all event channels is static.

This XML based description is part of the proposed application development process and Section 4 gives some example code for clarification.

3.2 Scheduling

HIJA uses a rate monotonic scheduling approach [8] with fixed priorities for all active objects in the local Java VM. Pre-emptive priority-based scheduling and the use of ceiling priority inheritance allows static calculation of thread response time and feasibility analysis.

The asynchronous event service is integrated into this mechanism and provides additional information for the scheduling analysis. To support periodic event transmission schemes, the parameters given in the event channel definition are used to arrange the highest priorities for the receiver threads, depending on buffer capacity in the CNI and release periods. Threads to handle sporadic events follow the algorithm for sporadic servers [25]. Also, for integration in response time analysis, a thread with highest priority is created.

example	release	deadline	priority
receiver S	sporadic	./.	highest
receiver 1	periodic	./.	highest a)
receiver 2	periodic	./.	highest a)
manager	periodic	./.	higher
handle	waiting	short	high b)
handle	waiting	medium	medium b)
handle	waiting	long	low b)

a) depending on socket buffer size and releases of channels on this network
b) deadline monotonic assigned

Table 1: Schedulable objects.

Table 1 illustrates periodic and sporadic threads resulting out of the simple event channel example in the next section. The calculated priorities are used for the feasibility analysis with all schedulable objects in the system.

4 Application Development

Built on the software pattern and required scheduling policy introduced in the last section, this section recommends a software design and implementation process to improve the development of high-integrity applications with hard real-time guarantees.

The requirement for static analysis of hard real-time systems results in the possibility of describing a distributed system with all parameters. The system execution depends on hardware profiles described for each node containing socket descriptions with buffer capacity, as well as hardware specification data for processor and the timing guarantees of the Java VM.

```
<node name=”system”>
 <hw-spec-data>uri<hw-spec-data>
 <socket name=”ttp”>
  <code>TTPSocket.instance()</code>
  <buffer>200</buffer>
 </socket>
```

Each node can be described by its server and client components. Servers describe event channels and possible events with static sizes and clients characterize handlers with their worst case computation time.

```
<server name =''send''>
 <channel id=''13'' name=''status''>
  <periodic>100ms</periodic>
  <deadline>80ms</deadline>
  <event id=''1'' name=''A''>
   <size>120</size>
  </event>
 </channel>
 <channel id=''35'' name=''work''>
  <periodic>200ms</periodic>
  <deadline>100ms</deadline>
 </channel>
 <channel id=''42'' name=''emergency''>
  <sporadic>500ms</sporadic>
  <deadline>50ms</deadline>
 </channel>
</server>
<client name=''rec''>
 <handler name=''control''>
  <event ref=''system.send.status.A'' />
  <code>CtrlAct.instance()</code>
  <wcct>50ms</wcct>
 </handler>
```

The XML fragments are reduced to the important parts. Also, necessary cross references to channels or events, as well as duplicate concepts, are omitted. The introduced XML notation is used for an event service node to set up the required active objects for receiving and handling events. Further active objects can be part of the handling logic or the rest of the program code.

5 Concluding Remarks and Future Work

The presented object-oriented event service framework facilitates the use of conventional COTS hardware and software components with asynchronous communication in high-integrity systems with hard real-time requirements. With static declaration of all system nodes it supports the programmer in the use of next generation programming techniques to create dependable and timely applications with low failure probability. The proposed direct publish/subscribe communication fits well in the development of high-integrity and scalable application scenarios on top of standard broadcast oriented networks and fieldbuses in embedded systems. Current development of the event service in HIJA is done for a static version, that is presented here, and a more dynamic version, for mixed soft and hard real-time requirements.

References

[1] ARINC. *ARINC Standard. 629 Part 1-5 - Multi-Transmitter Data Bus, Part 1-Technical Description*, 1999. http://www.arinc.com.

[2] Belschner, R. et al. *FlexRay. Requirements Specification*, 2.0.2 ed., April 2002.

[3] Brinkschulte, U. et al. Distributed Real-Time Computing for Microcontrollers - the OSA+ Approach. In *Proceedings of the International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2002)* (2002).

[4] Condor Engineering, Inc. *AFDX / ARINC 664 Tutorial (1500-049)*, 1.0 ed. Santa Barbara, CA 93101, November 2004.

[5] Eggermont, L. D. J. Embedded Systems Roadmap 2002, March 2002.

[6] Eugster, P. Th. et al. The Many Faces of Publish/Subscribe. *ACM Computing Surveys 35*, 2 (June 2003), 114–131.

[7] HIJA. High Integrity Java Applications. Project Website. http://www.hija.info, 2004.

[8] Klein, M. H. *A Practitioner's Handbook for Real-Time Analysis. Guide to Rate Monotonic Analysis for Real-Time Systems.* Kluwer Academic Publishers, 1993.

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

[10] level Group on Embedded Systems, H. Building artemis. report. Luxembourg: Office for Official Publications of the European Communities, 2004.

[11] Object Management Group, Inc. *CORBA Messaging*, orbos/98-05-06 ed., May 1998.

[12] Object Management Group, Inc. *Minimum CORBA Specification*, version 1.0, formal/02-08-01 ed., August 2002.

[13] Object Management Group, Inc. *RealTime - CORBA Specification (Dynamic Scheduling)*, version 2.0, formal/03-11-01 ed., November 2003.

[14] Object Management Group, Inc. *Common Object Request Broker Architecture: Core Specification*, version 3.0.3 - editorial changes, formal/04-03-01 ed., March 2004.

[15] Object Management Group, Inc. *CORBA Event Service Specification*, 1.2 ed., October 2004.

[16] Object Management Group, Inc. *CORBA Notification Service Specification*, version 1.1, formal/04-10-11 ed., October 2004.

[17] Object Management Group, Inc. *Real-Time - CORBA Specification (Static Scheduling)*, version 1.2, formal/05-01-04 ed., January 2005.

[18] Project Partners. D8.1 - HIJA White Paper. Tech. rep., The Open Group, June 2005.

[19] Puschner, P. et al. A Profile for High-Integrity Real-Time Java Programs. In *Proceedings of the 4th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC)* (2001).

[20] Robert Bosch GmbH. *CAN Specification*, 2.0 ed. Postfach 50, D-7000 Stuttgart 1, September 1991.

[21] Rose, M. T. *The Open Book. A Practical Perspecive on OSI.* Prentice Hall, 1989.

[22] Schanne, M. Anforderungsanalyse für asynchrone Kommunikation beim Entwurf von objektorientierten, verteilten Systemen unter Echtzeitbedingungen. In *Proceedings of Workshop Zuverlässigkeit in eingebetteten Systemen* (October 2005). to appear.

[23] Schanne, M. Real-Time Communication with a Receiver Collective, Activity Manager, and Queues. In *Proceedings of IADIS Internatonal Conference Applied Computing 2005* (2005).

[24] Schmidt, D. C. et al. *Pattern-oriented Software Architecture. Patterns for Concurrent and Networked Objects*, vol. 2. John Wiley & Sons Ltd., April 2001.

[25] Sprunt, B. et al. Aperiodic Task Scheduling for Hard-Real-Time Systems. *The Journal of Real-Time Systems 1* (1989), 27–60.

[26] Tanenbaum, A. S., and van Steen, M. *Distributed Systems. Principles and Paradigms.* Prentice Hall, Inc, 2002.

[27] Technologies, I. S. ARTIST Year 2 Roadmap. http://www.artist-embedded.org, May 2004.

[28] Thomesse, J. P. A review of the fieldbuses. *Annual Reviews in Control 22* (1998), 35–45.

[29] TTA Group. *TTP. Time-Trigered Protocol TTP/C. High-Level Specification Document*, July 2002.

[30] Wellings, A. J. et al. A Framework for Integrating the Real-Time Specification for Java and Java's Remote Method Invocation. In *Proceedings of the 5th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing* (2002), pp. 13–22.