REAL-TIME COMMUNICATION WITH A RECEIVER COLLECTIVE, ACTIVITY MANAGER, AND QUEUES

Marc Schanne
Software Engineering
FZI Forschungszentrum Informatik
schanne@fzi.de

ABSTRACT

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 applications in automotive, avionics, telecommunications, and process control.
This short paper discusses the advantages of an asynchronous push event notification mechanism in the software engineering for object-oriented, efficient, and deterministic real-time applications, and presents an architectural design pattern for the proposed decentralized publish/subscribe programming model, with a logical event channel as central component in this pattern.

KEYWORDS

Real-Time, Event Service, Networking, Embedded Systems

1. INTRODUCTION

This short paper presents the ongoing research work for a network model in High-Integrity Java Applications[1] (HIJA) [1]. The analysis of the networking support for Architecturally Neutral, high-integrity Real-Time Systems (ANRTS) based on the Real-Time Specification for Java (RTSJ) results in the need for an additional asynchronous communication model besides real-time remote method invocations (RT-RMI). In addition to synchronous communication like RT-CORBA [2] or RT-RMI [3] a direct publish/subscribe event service offers several benefits to distributed real-time systems. The communication is direct without any intermediate active component. For a brief discussion of the assets and the main drawback of this communication in real-time systems refer to section 2.

The event channel network [4] utilizes a simple broadcast oriented socket interface for byte array access to any underlying network system. The event channel network design can be illustrated

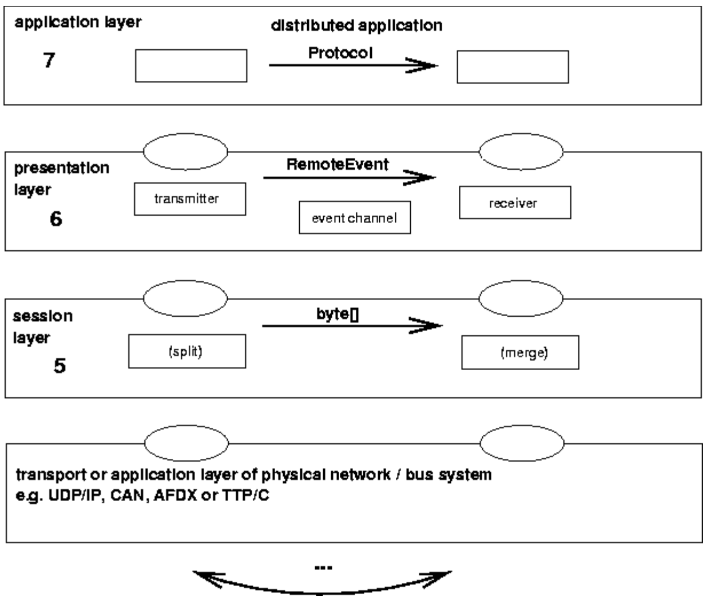

Figure 1: Event channel network application (layer structure)

1 IST-511718, research project in the 6th framework program of the European Commission

with reference to the ISO/OSI layer structure (see figure 1).

On top of transport functionality of the underlying network system the event channel network implements a many-to-many event dissemination. The session layer handles data as byte arrays and provides a simple object serialization to encode user data and protocol information, while the presentation layer establishes the concept of events and event channels. Event channels and event types are encoded as integers to support an easy implementation on arbitrary networks. Further details of the event channel declaration and usage of the event channel network is discussed in section 3.

For hard real-time use a network system like AFDX[2] [5] or TTP[3] [6] with predictable timeliness and bandwidth for bounding the delivery time of events is necessary. Whereas that system configuration is necessary under hard real-time constraints the event channel network is also useful in soft real-time environments. With a more lax declaration of system parameters even an integration with non real-time communication components, e.g. off-board control units, is possible. The paper closes with the description of the dynamic feasibilities (see section 5) to enhance business critical systems with soft real-time warranties for a better quality of service (QoS).

2. ASSETS AND DRAWBACK

To motivate the use of a publish/subscribe event service this section discusses several benefits for distributed real-time systems, and opposes problems with predictability in asynchronous communication. For a introduction of the concepts of publish/subscribe the article [7] provides a comparison and classification of the many faces of publish/subscribe in established interaction schemes. The adaptation for real-time systems is well founded:

- Anonymous communications: Publishers do not need to know the addresses of subscribers and similarly subscribers do not need to know the data publisher 's identity;
- Decoupling of publishers and subscribers: Flexibility and simplified code reuse;
- Many-to-many communication: Utilization of broadcast-oriented field busses in the area of embedded systems;
- Scalability: Easily expandable asynchronous coordination without blocking request-response, scales well from the most basic to very large systems;
- Simplified programming of communication in automation devices: The information does not need to be read or pulled, and there is no need to know internal memory maps of databases structure of unknown devices;
- Efficient use of network bandwidth: No request traffic, use of direct event-driven transfer;
- Robust application design: Support for fail-over and migration;
- Portability across platforms: While the system design is well implemented on top of the RTSJ, the communication protocol itself and the system architecture is platform and network independent;
- Maps well to real-time needs: Use for streaming signals, status updates, event-driven commands in real-time applications.

To use these system advantages in software design and development for embedded systems the lack of synchronization is the obvious drawback in reliable real-time programs. The event channel network has to manage this with binding asynchronous communication to timeliness in interaction and scheduleability.

3. EVENT CHANNEL

The event channel, depicted in figure 2, is the central concept in the event channel network. This logical object is used to group events transmitted and received over the underlying network system. The encoding via an integer is used to provide the topic-based publish/subscribe service. The event channel defines a list of

2 Avionics Full Duplex Switched Ethernet, real-time network protocol standardized as ARINC664

3 Time Triggered Protocol, a class of time-triggered real-time bussystems to meet high safety and fault tolerance requirements

accepted events and requirements for the transmitting, receiving, and handling threads, processing these events. The parameters contain basic information like necessary queuing capacities, computation resources, adequate priorities, scheduleability settings, and required communication bandwidth for each node, which can be used for statical analysis, and verifycation of the real-time capabilities against the hardware and network configuration.

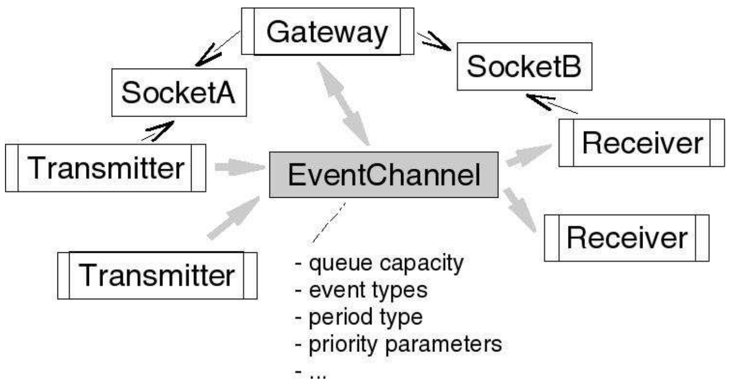

Figure 2: Event Channel in a direct publish/subscribe system with active objects

Network management is distributed. Each node has its own managing unit, thereby avoiding a centralized point of failure. The application has access to the event channels through sockets to the underlying network. Active objects like receiver, and transmitter have to satisfy the described requirements and provide an application logic independent use of the event channel network. Each event channel can be connected to multiple underlying networks, in the joined application nodes. The events are directly transferred without any centralized network management component. For each network socket one receiver thread is responsible. These threads dispatch received events, and forward them to an activity manager[4] through priority queues. Another transmitter thread coordinates the access to the underlying networks. To provide a network gateway between different physical segments an additional active component, a gateway component with socket access to both physical networks, has to be defined. Figure 2 further shows the association between these components and the data flow between transmitter and receiver in different physical networks connected through a gateway. This gateway is like the transmitter or receiver an active object, and scheduleable thread. All these threads run concurrently in the virtual machine and need a predefined scheduling behavior. For more information see the next section.

The event channel itself is no active object and provides no further functionality for the decentralized publish/subscribe event service. The communication protocol is directly built on top of the functionality of the physical network or bus and the light-weight Java implementation integrates the required threads in the scheduling of the local RTSJ compliant virtual machines[5]. The event channel network provides a network independent topic-based publish/subscribe service and all activity is located in the nodes of the distributed embedded system.

4. SCHEDULING

The event channel definition is used to specify requirements for active objects of an application using the event channel network. With a pre-emptive priority-based scheduling mechanism the necessary threads allow concurrent execution and event transfer in real-time. For scheduleability either periodic or sporadic actions with guaranteed minimum inter-arrival times[6] are allowed, and as suggested in [8] for a clear design and overall predictability a segmentation of the real-time application in initialization and mission phase is required. The following subsections clarify this logical segmentation and the subsection 4.1 mainly describes the required scheduling behavior in the mission phase. The restriction to periodic and sporadic threads is used to predict the resource requirements of the threads in the event channel network and effects on periodic and sporadic event communication. In the dynamic case of non real-time systems these parameters are used

4 One thread per virtual machine

5 HIJA proposes a high-integrity subset of the RTSJ model

6 Aperiodic actions without this warranty are not allowed in the profiles for hard and soft real-time applications defined with HIJA

to establish a failure model with exception handling in the application layer (see section 5).

4.1 Priorities

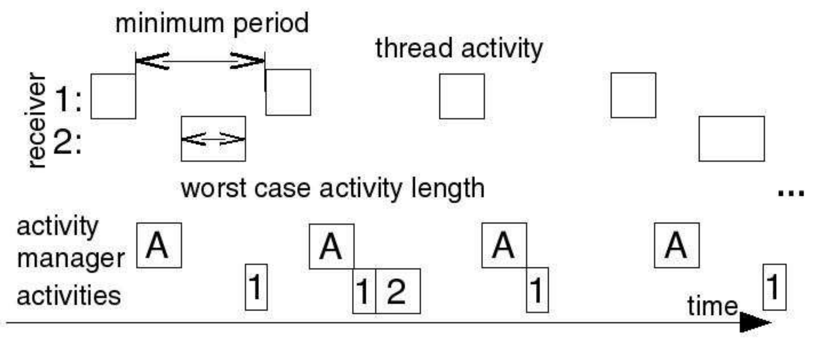

Figure 3: Scheduling example with two receiver threads

To enable the event channel network for hard real-time, a fixed priority scheduling (e.g. deadline monotonic or rate monotonic analysis [9]) is necessary. The runtime system has to implement the priority ceiling emulation protocol to avoid deadlocks. The schedule-able objects accessing the network sockets are configured with the highest available priority to guarantee a blocking-free communication.

The collective of receivers is forwarding the events to the activity manager through queues, sorted by priority. The activity manager thread activates adequate waiting threads (activities) to execute registered handlers.

Requirements of computation resources, availability of queuing capacities, and specification of deadlines and timeliness allow the schedule verification with a fixed priority scheduling algorithm. This prediction uses the minimum period[7] of the required thread activities to be scheduled. The worst case execution time of all activities is required to verify. Whether the threads are scheduleable the whole system has to return in an initial state after the least common multiple of all periods.

To demultiplex and dispatch the incoming events the activity manager thread is used, its priority is smaller than the priority of the collective and is responsible to manage a pool of activity threads, offering necessary computation resources for the application event handlers. Figure 3 depicts a small example with a collective of two receiver threads and illustrates the relationship between the period type, required computation resources and available queuing capacities. The activity manager is used to receive events from the first-in-first-out (fifo) priority queue collection, and forward them to a waiting activity thread to handle.

4.2 Initialization Phase

The initialization phase is used for non-time-critical activities like creation and configuration of all mission real-time threads, memory objects, and event handlers. It is assumed no mode changes in the application follow. While in hard real-time applications no dynamic negotiation about event channels, event types, or transfer bandwidth is possible all information have to be predicted and can be statically verified. The initialization phase makes use of the basic administration protocol of the event channel network. For each communication path the event channel description has to be distributed among the joined application nodes, and related handlers have to be registered.

4.3 Mission Phase

In the mission phase the event communication takes place. While the event channel network does not depend on special asynchronous network capabilities, the receiver collective is designed similar to the reactor pattern [10]. The proposed design pattern of the event channel network provides equivalent synchronous event demultiplexer, but extends it with an additional activity manager and asynchronous queue-interaction, for a better partition of fast reception and efficient handling of asynchronous events.

7 This interval is bound to the periodic and sporadic event receptions of the receiver collective.

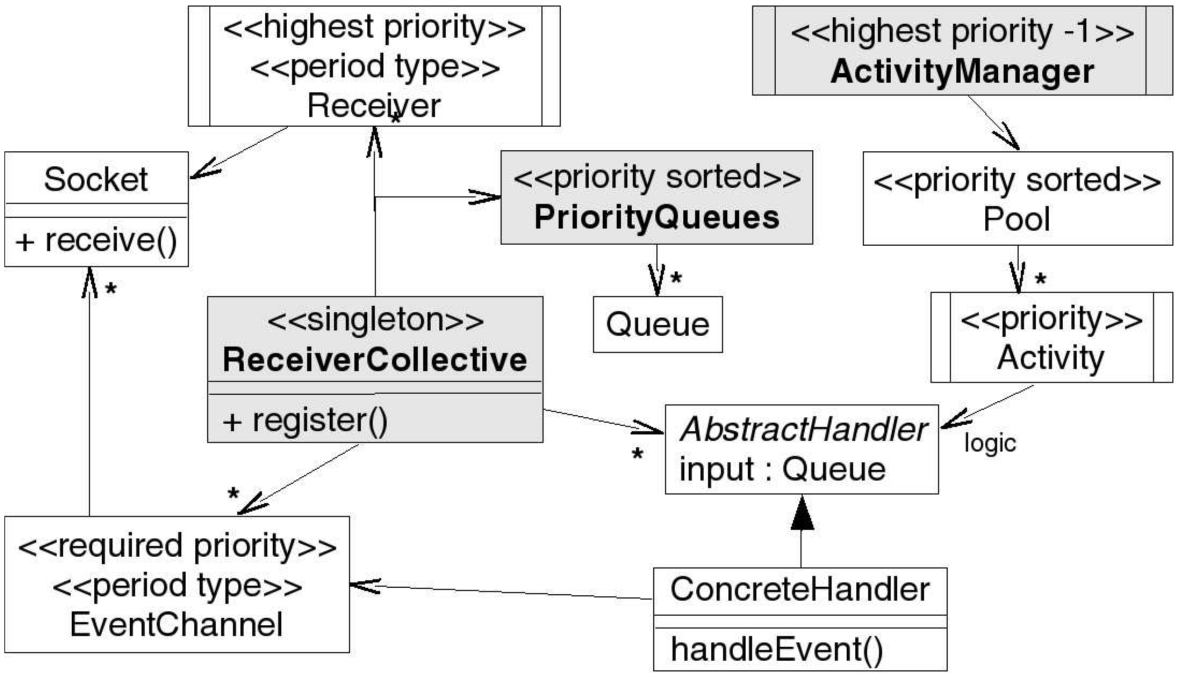

Figure 4: Overview of the main system components

This event service design is depicted in figure 4. The receiver collective is implemented as singleton on each node and is used to organize the infrastructure of receivers for each associated socket, available event channels, required fifo queues, and registration entry point for local event handlers. Each receiver has access to this configuration and forwards received events in his action period to the activity manager.

The activity manager controls a pool of activity threads with different priorities and controls periodically the asynchronous received events in the queues and put them into the input queue of waiting activity threads for immediate handling.
In hard real-time systems the introduced architectural design pattern guarantees demultiplexing of asynchronous arriving events from different sources, that are bound to deadlines described in the event channel configuration. For the use with non-hard real-time systems the mechanism with periodic receivers and activity manager enables an efficient event dispatching with pooled threads for execution under dependable priorities.

5. DYNAMICS

With dynamic features for discovery, creation, and advertisement of nodes and event channels the event channel network is useful in soft real-time applications. These dynamic aspects allows a more general implementation of the direct publish/subscribe event service with exception handling. For non real-time dynamic environments even an implementation on top of the UDP/IP protocol is applicable. By defining a failure model with exceptions for unexpected or sub-optimal behavior an overall system health can be measured.

The two main models of communication: the receiver monitoring and transmitter monitoring are based on periodic and sporadic event types. This allows a failure handling in the application layer. The application must know how frequently events should arrive, and take appropriate actions when they do not. For further discussion of the possible failure modes and error handling refer to [4], where a soundly analysis for UDP/IP, Controller Area Network (CAN) and TTP/C[8] for soft real-time feasibility can be found.

In hard real-time systems this technique is not suitable and any failure results in a total error of the event

8 An implementation of the Time Triggered Protocol [5]

channel network and the application.

6. CONCLUSION

This short overview identifies the benefits of asynchronous, indirect many-to-many communication in the area of networked, real-time embedded applications and proposes an easy to implement Java and RTSJ based design pattern for architecturally neutral, high-integrity real-time systems. The event channel network itself is not designed to implement network reliability with any protocol over-head, rather it utilizes the full features of the underlying (real-time) networks or busses.

The use of a synchronous network interface for an asynchronous communication scheme in real-time is possible and allows an easy to understand topic-based publish/subscribe interaction. The communication is direct without any central management unit and each node is responsible for its own administration. The distributed and decentralized architecture avoids any single point of failure and is excellently qualified for integration of real-time systems within non real-time environments.

REFERENCES

[1] HIJA – High-Integrity Java applications. *Project Website.* http://www.hija.info

[2] Object Management Group, Inc., 2002. *Real-Time CORBA Specification*. Version 1.1.

[3] Wellings, A. et al, 2002. 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*, pp 13-22

[4] Schanne, M. and Hunt J., 2004. Remote event service design. *Technical Report.* FZI Forschungszentrum Informatik, Karlsruhe, Germany. Deliverable D4.2 describing the HIDOORS event channel network.

[5] Condor Engineering, Inc., 2004. *AFDX/ARINC 664 Tutorial (1500-049).* Version 1.0.

[6] TTA-Group, 2002. *Time-Triggered Protocol TTP/C. High-Level Specification Document.* Version 1.0.0.

[7] Eugster, P. Th. et al, 2003. The Many Faces of Publish/Subscribe. *In ACM Computing Surveys Journal.* Vol. 35, No. 2, pp 114-131.

[8] Puschner, P. and Wellings A., 2001. 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).*

[9] Klein, M. et al, 1999. *A Practitioner's handbook for real time analysis : guide to rate monotonic analysis for real-time systems.*

[10] Schmidt, D. 1994. Reactor. An Object Behavioral Pattern for Demultiplexing and Dispatching Handles for Synchronous Events. *In Proceedings of the First Pattern Languages of Programs Conference.*