Integrated Development of Distributed Real-Time Applications with Asynchronous Communication

Marc Schanne
FZI Forschungszentrum Informatik
Research Center for Information Technologies
Software Engineering (SE)
Haid-und-Neu-Str. 10-14
76131 Karlsruhe, Germany
schanne@fzi.de

ABSTRACT

This paper illustrates added values of an integrated development methodology for distributed real-time applications. Strategic focuses of the proposed methodology and middleware framework are real-time and embedded applications. The framework builds on top of real-time Java with an asynchronous communication model for broadcast networks and fieldbusses. Results of the EC research projects HIDOORS (IST 2001-32329) and HIJA (IST 2003-511718) are used to establish a methodology which supports design, implementation, and analysis of distributed real-time applications.

Keywords

Distributed Real-Time, Integrated Development

1. INTRODUCTION

Distributed high-integrity systems pervade daily life and are gaining importance. Their failure can cause loss of life, environmental harm, or significant financial loss. The *EventCannelNetwork* (ECN [3]) has been developed to increase dependability and reusability for these systems. ECN supports the development of distributed real-time applications for embedded systems with real-time requirements.
Integrated development is supported by the ECN middleware framework and the ECN development methodology. The methodology allows the development of distributed systems with loosely coupling of networked system components and services. The use of an object-oriented component design supports code reuse and the methodology integrates declarative application design with code generation for implementation and static analysis to reduce testing efforts.
An asynchronous communication model with a direct publish/subscribe event service is used [17]. The ECN middleware allows the development of distributed applications without a central administration unit or persistence mechanism. Components in each communication node are responsible to guarantee timely event reception and handling. The ECN middleware utilizes a simple broadcast oriented socket interface for byte array access to any underlying network system[1]. For hard real-time use a network system like AFDX[2] [8] or TTP[3] [10, 22] with deterministic timeliness and bandwidth for bounding the delivery time of events is necessary.
The ECN middleware framework with asynchronous event communication and object-oriented component design for distributed systems was developed in two research projects founded by the European Commission. Both projects addressed the use of the Real-Time Specification for Java (RTSJ) with embedded systems. HIDOORS ("High Integrity Distributed Object-Oriented Real-time Systems", IST 2001-32329 [1]) introduced the ECN middleware for the interaction between distributed components, and HIJA ("High Integrity Java Applications", IST 2003-511718 [2]) extended this for the use with both flexible and hard real-time requirements. Experiences in application development influenced and es-

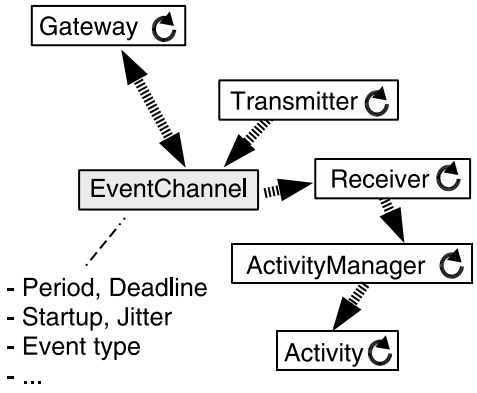

Figure 1: Components in the ECN middleware

JTRES'07, September 26–28, 2007, Vienna, Austria.

[1]Fieldbusses – common in embedded systems – are supported with the interface of an ISO/OSI transport layer.
[2]Avionics Full Duplex Switched Ethernet, real-time network protocol standardized as ARINC 664.
[3]Time Triggered Protocol, a class of time-triggered real-time bussystems to meet high safety and fault tolerance requirements.

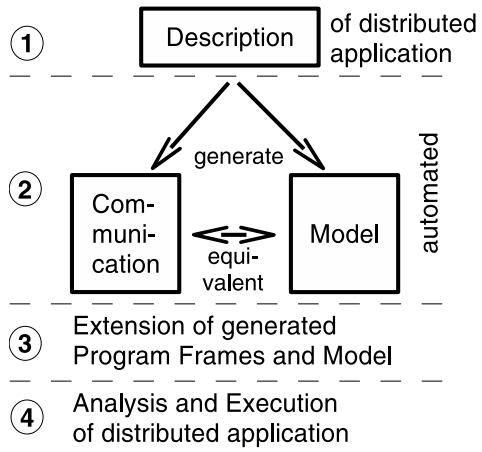

Figure 2: ECN methodology

tablished an ECN methodology, which supports design, implementation, and analysis of distributed real-time applications.

2. METHODOLOGY FOR INTEGRATED DEVELOPMENT

The ECN methodology uses the known structure of distributed applications with event communication. An integrated development with application design, code generation and static execution analysis is supported thereby. This section introduces four major steps of the ECN methodology for integrated development. Figure 2 shows an abstract view of these steps. The following descriptions classify them in a common software development process:

(1) Initial step of the development of the distributed application is the declaration of nodes and event channels in the design phase. The central concept in ECN middleware based distributed applications is the use of event channel objects. Figure 1 introduces the class of these logical components in the center of other middleware components. Events are grouped with event channels and the developer has to declare communication characteristics through XML descriptions for each event channel at each node.
It is necessary to identify communication links and sockets for the nodes in the distributed system. Communication links (i.e., event channels) abstract from physical network connections and protocols, whereas sockets (i.e., physical network access with a transport

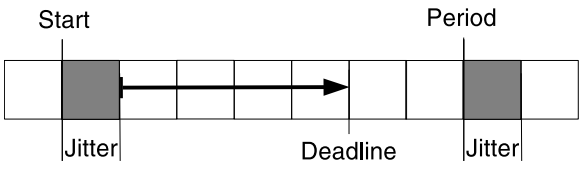

Figure 3: Communication attributes of example event channel

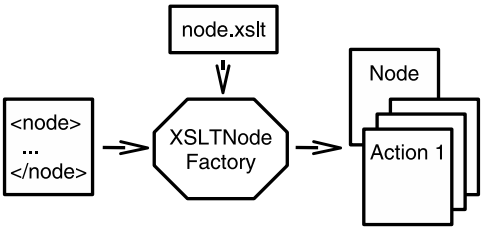

Figure 4: XSLT code generation

layer) map this design to concrete hardware systems [13]. In the design phase Januar [14] emphasizes the need of support to control the development of distributed applications. His experience results in an Eclipse Plug-in for application development. The use of such an assistant is well advised to avoid error-prone "copy-n-paste" from a sample or a former project.
The declaration of communication attributes (e.g., period or deadline) predicates the following steps in the development process. Figure 3 depicts these attributes for an example event channel. Each node registered to an event channel has to keep the deadline for processing events of this channel. To meet the requirements of several concurrent event channels the ECN middleware separates event handling from event reception. The reception is part of the middleware and values for worst case execution time (WCET) have to be analysed for each receiving node — hardware and Java runtime environment.

(2) The second step is automated. The implementation phase is supported with generation of standard program code for communication. A diagram of the XSL transformation in the prototype implementation for the HIDOORS project is presented in figure 4. An equivalent analysis model is generated here, too. The ECN middleware uses a design pattern with a receiver collective, activity manager, and queues [16]. Event reception is separated from event handling, and the middleware is responsible to provide real-time threads with appropriate priorities to guarantee the processing in necessary deadlines. These deadlines for concurrent event handling of different event channels are assigned in a deadline monotonic [7] order.
Communication attributes for periodic or sporadic event arrival are declared in design phase. The program generator uses these attributes to calculate thread parameters in each node. Section 3 explains this more detailed.
The generation allows creating equivalent model descriptions for a static analysis by nodes. It is necessary to calculate WCET bounds for basic activities (i.e., queue read/write and copy actions, as well as any thread notification action) in the ECN middleware API for the used hardware and runtime system[4].
Step four uses this description for a thread scheduling analysis in each node. Event reception and handling with real-time requirements become feasible.

[4]The use of tools for WCET analysis are recommended, but if the calculation of upper bounds is not available, the software developer has to provide estimations for these costs.

(3) Step three allows the developer to implement application specific program code. Generated empty frames for application logic with the ECN middleware have to be filled. The methodology guides the implementation phase with components for event handling. Standard factory methods or constructors are used for an easy integration in generated program code of communication and real-time resource provision. With WCET analysis upper bounds for these code fragments have to be calculated. These bounds for application specific event handling are also part of the analysis model for the static analysis in step four.
Application logic beyond standard event handling can be provided in new program components as well. The analysis model has to be completed with additional threads or shared resources.

(4) For hard real-time requirements the development process is extended in step four of the methodology. The use of static analysis methods to reduce testing efforts are more and more accepted in academic, industrial and governmental organizations. It is possible to analyse the behaviour of dynamic systems before execution. HIJA identified the need of fixed priority scheduling with RTSJ for hard real-time systems [21]. Fixed priorities allow the static analysis for all threads and due to the results of WCET analysis a scheduling analysis with verification of declared deadlines is possible. As a result of the HIJA research project the schedulability and real-time analysis suite MAST [4, 9] is used for this analysis step.
While a schedulability analysis of the distributed system by node can not verify end-to-end communication requirements, the methodology requires a hardware and runtime system with communication network for adequate real-time, bandwidth and capacity guarantees. The ECN middleware uses these features and the proposed methodology enables a simplified development of distributed applications with asynchronous event communication for broadcast networks [18]. With static analysis the methodology provides an alternative to extended runtime tests. Requirements specifications like the DO-178B (Software Considerations in Airbone Systems and Equipment Certification) of the RTCA ("Radio Technical Commission for Aeronautics" [5]) in the U.S. are standard in the avionics software development. With support for new analysis methods it is possible to satisfy these requirements with less expensive tests.

The ECN methodology is a result of the software development experience in the research projects HIDOORS and HIJA. It focuses on the integrated support of the developer. The goal is a controlled and well guided process to improve the development of safety critical distributed applications.
While asynchronous communication is not extensively used in distributed systems with real-time requirements, this communication mechanism fits well to an object-oriented and component based software design. The ECN middleware provides a decentralized communication infrastructure and XML files describe the distributed system with real-time requirements in each node. These descriptions are used to configure the middleware components for each node and with hard real-time requirements the code generation of necessary buffer objects and real-time threads is possible.
Development by description is comparable to system development with the Unified Modeling Language (UML) and a model driven software development process. UML 2.0 supports additional profiles and the "UML Profile for Schedulability, Performance and Time" (UML-RT) enables the design of object-oriented systems with real-time requirements [11]. In the HIDOORS project the XML description was used as starting point for an implementation of the UML-RT with the ECN middleware [19].
Even though the UML and "Model Driven Architecture" (MDA) propose a straightforward model transformation approach. An integrated development with automated model transformation is not available yet. Software development with less complex domain specific languages (DSL) and domain specific modeling (DSM) become popular in recent times. Industrial experiences of DSM consistently show it to be 5-10 times faster than current practices, including current UML-based implementations of MDA [6, 20]. The ECN methodology and middleware follow a similar concept. The focus on embedded real-time systems with broadcast networks allows the integrated development of distributed applications. The methodology supports application design, implementation, and static analysis.

3. ASYNCHRONOUS MIDDLEWARE FRAMEWORK

The ECN methodology builds on a middleware framework supporting asynchronous event communication with synchronous COTS components. Event channels declare necessary deadlines for event handling after event arrival in the nodes of the distributed system. The ECN middleware requires event transmission with upper bounds for this guarantee. Figure 3 illustrates an example event channel declaration with all necessary time frames to receive an event.
Start declares the first possible point in time where an event can arrive. With *period* the time frame for periodical recurrence of events in this channel is given, and *jitter* describes the possible inaccuracy of these arrivals. After an event arrival at a node *deadline* declares the time until this node has to receive and handle the event successfully.
The grey boxes represent time frames, where the receiving node needs to be able to read arriving events from the input queue of the physical network interface. The processor has to provide processing capacity at any point of time in these frames. A variance of jitter is possible here. In this time frame the processor has to be ready to execute the receiver thread. This is done by fixed priority scheduling and highest priority for this thread. While no event is available, the receiver thread blocks and other lower priority threads can be executed. This mechanism allows the asynchronous event handling with synchronous networks and network interfaces.
The figure 5 shows a sequence diagram for the event reception and handling for the example event channel described in figure 3. The interactions between threads of the ECN middleware are presented. The used thread schedule with changes in this schedule is described for the following steps.

(1) Arrival of an event in the input queue of the network interface.

(2) Activation of the blocked receiver thread.

(3) Notification of the activity manager thread.

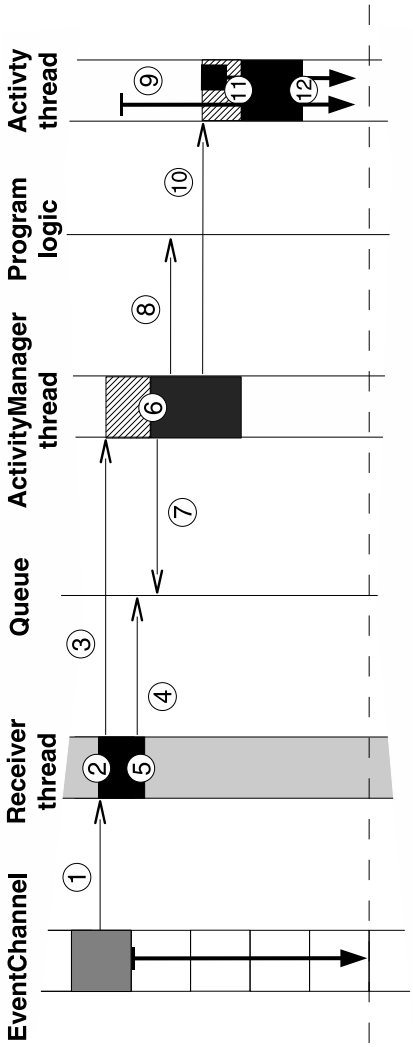

Figure 5: Sequence diagram for successful event reception and handling

(4) Receive the event: Copy attribute values in buffer object of the queuing system.

(5) Waiting (blocking read) of the receiver thread until next event arrival.

(6) Activation of the activity manager thread.

(7) Dispatching of the received event.

(8) Copy event (attribute values) to the input queue of the event handling activity thread.

(9) Recalculation of the activity thread deadline depending on a reception time stamp.

(10) Change to a new thread schedule.

(11) Activation of the activity thread and handling of the event.

(12) Successful reception and handling of the event in the deadline given with the event channel.

The ECN middleware controls the reception of events. Each node has to provide processing capacity and threads to receive and handle asynchronous arriving events in their deadlines. The design of the distributed application declares time frames with bound jitters for all event arrivals. Real-time requirements are declared with deadlines for successful handling.

After arrival of a data package in the network interface the middleware has to decode these data. A receiver thread, responsible for this network socket, reads the input queue of the network interface and creates a Java object representing the event for further processing.

The activity manager thread is responsible for dispatching pending event objects. Because the thread has a lower priority than any receiver thread (i.e., highest available priority minus 1), the execution is postponed until all receiver threads are in a blocked state again.

All events are saved in a priority-sorted queuing system for further handling. Existing buffer objects are reused to support hard real-time requirements, thus no object creation is supported in the mission phase of an application. Receiver and activity manager threads copy attributes to members in reusable Java objects.

Receiver threads[5] work with highest priority to guarantee the reception of all available events. Each receiving node has a thread schedule depending on event channel communication attributes (periodic or aperiodic with a minimum inter-arrival time) that allows the reception of all interesting events. When all receivers are in blocked state, the management and event handling has to be done. Static analysis for each node can verify, that no processor is in danger to be overloaded. If a node processor requires too much processing capacity to receive and handle required event channels, a system redesign is necessary.

The activity manager thread reads pending events from the queuing system, identifies their types and required handler actions. It copies the event attributes into the input buffer object for this handler logic and prepares an activity thread with appropriate priority for execution. Events with short deadlines are handled by threads with high priorities [16]. This deadline monotonic approach with a fixed priority scheduler allows meeting all deadlines. The use of fixed priorities also allows a static analysis of the thread schedule [21] in each communication node.

After event arrival the receiver thread copied event data into a reused buffer object of the queuing system. The point in time of the reception was saved as a time stamp. For asynchronous event handling in real-time restrictions this information is used for recalculation the deadline of the activity thread. After reception the ECN middleware guarantees handling bound by a declared deadline of the event channel. This timing requirement is part of the channel description. The recalculation of thread attributes like deadline changes are always conservative (i.e., the recalculation with the time stamp shortens the deadline), the schedule of the threads needs no new online scheduling/feasibility analysis. The runtime system is only responsible to control the compliance of these thread parameters and for example deadline missed handlers get fired. In flexible real-time[6] systems this feature is used to react on exceptional states.

[5] One thread for each used network system or fieldbus.

[6] In the HIJA terminology this is used for systems with soft or mixed (hard and soft) real-time requirements.

After assigning events to their handler activity threads, the activity manager thread's work is done. Another thread with the largest available priority is activated. In the example sequence the activity thread to handle the received event is executed and the event handling is done.
For the example event channel the event handling has finished within the declared deadline. For systems with hard real-time requirements the ECN methodology supports a static worst case analysis of these requirements (i.e., deadlines). An analysis model representing real-time threads and shared resources of the ECN middleware is generated, and the developer extends this model for schedulability analysis with application specific logic.
The ECN middleware presented in the previous description and the sequence diagram in figure 5 provides a sound architecture design pattern to support the proposed methodology. The ECN middleware implements this pattern with separation of reception and handling, and distributed systems are composed (i.e., declared) with applications specific logic and generated standard code for communication.
Because asynchronous event communication with synchronous network interfaces and standard communication hardware requires concurrent processing of events, a multithreaded runtime systems is necessary. The Java platform for safety critical embedded systems (JSR302 [15]) or developments within the EC research project HIJA are reasonable.

4. CONCLUSIONS

The use of asynchronous event communication with real-time and embedded systems provides several benefits for distributed applications [17]. Decoupling of senders and receivers in the proposed publish/subscribe system allows independent development and reuse of components. The composition by declarative programming supports scalable and robust application design. In embedded systems with broadcast oriented communication networks an efficient use of bandwidth and processing capacities over distributed nodes is possible.
The development of distributed applications is improved with a declarative design method, the implementation by generation of standard communication and real-time management code, as well as created analysis models equivalent to program code. The ECN middleware for real-time Java in embedded systems implements a design pattern to support concurrent and timely handling of several periodic or sporadic events[7]. Real-time requirements for event handling are declared for each channel in a system and the compliance of the implementation (i.e., software and hardware) with these requirements is verified for each node of the application. This use of a domain related abstraction for event channels proposes a standard for the description of distributed embedded real-time systems with asynchronous event communication, and an integrated software development is supported by the ECN methodology.

5. ACKNOWLEDGMENTS

This work has been supported by the European Commission with funding for the research projects HIDOORS (IST 2001-32329) and HIJA (IST 2003-511718). The author would like to thank colleagues participating in these projects for their help and comments on his work. Thanks also to the FZI and the University Karlsruhe for the possibility to work on a PhD with autonomous research on an interesting topic.

[7]Result of the HIJA research project are two implementations for hard and flexible real-time requirements.

6. REFERENCES

[1] HIDOORS. High Integrity Distributed Object-Oriented Realtime Systems. Project website. http://www.hidoors.org, 2002.
[2] HIJA. High Integrity Java Applications. Project website. http://www.hija.info, 2004.
[3] ECN. Event Channel Network. Project website. http://www.eventchannelnetwork.org, 2006.
[4] MAST. Modeling and Analysis Suite for Real-Time Applications. Project website. http://mast.unican.es, September 2005.
[5] RTCA. Radio Technical Commission for Aeronautics. Website. http://www.rtca.org, 2006.
[6] DSM Forum. What is Domain-Specific Modeling? http://www.dsmforum.org/why.html, December 2006.
[7] Neil C. Audsley, Alan Burns, M. F. Richardson, and Andy J. Wellings. Hard Real-Time Scheduling: The Deadline-Monotonic Approach. In *Proceedings of the 8th IEEE Workshop on Real-Time Operating Systems*, 1991.
[8] AEEC Airlines Electronic Engineering Committee. Circulation Prior to Adoption Consideration Draft 4 of ARINC Project Paper 664: Aircraft Data Network, Part 7 - Avionics Full Duplex Switched Ethernet (AFDX) Network. ARINC Aeronautical Radio, Inc., 2551 Riva Road, Annapolis, Maryland 21401-7465 USA, February 2005.
[9] Jose Maria Drake, Michael Gonzalez Harbour, Jose Javier Gutierrez, Patricia Lopez Martinez, Julio Luis Medina, and Jose Carlos Palencia. Modeling and Analysis Suite for Real Time Applications (MAST 1.3.6). Description of the MAST Model. Universidad de Cantabria, June 2004.
[10] Wilfried Elmenreich and Richard Ipp. Introduction to TTP/C and TTP/A. Technical report, Vienna University of Technology and TTTech ComputerTechnik, 2003.
[11] Susanne Graf, Ileana Ober, and Iulian Ober. A real-time profile for UML. *Software Tools for Technology Transfer (STTT) Journal*, November 2004.
[12] HIJA project partners. D7.4b - HIJA Methodology Handbook. Technical Report 1.4, The Open Group, September 2006.
[13] Dr. James J. Hunt, editor. *The HIDOORS Methodology. Using Java in Realtime and Embedded Systems.* aicas GmbH, Karlsruhe, Germany, 2004.
[14] Mohammad Athar Januar. Bewertung von Programmier- und Entwicklungsassistenten am Beispiel eines Eclipse Plug-Ins für den Entwurf verteilter Systeme mit der EventChannelNetwork-Kommunikationsinfrastruktur, 2006. Available over ECN project website [3].
[15] C. Douglass Locke. JSR 302: Safety Critical Java Technology. JSR Website http://jcp.org/en/jsr/detail?id=302, July 2006.
[16] Marc Schanne. Real-Time Communication with a Receiver Collective, Activity Manager, and Queues. In

Proceedings of IADIS International Conference Applied Computing 2005, 2005.

[17] Marc Schanne. Real-Time Communication with Direct Publish/Subscribe Event Service. In *Internal report 3rd Workshop on Java Technologies for Real-time and Embedded Systems (JTRES) OOPSLA 2005*, October 2005.

[18] Marc Schanne. *Methodik zur durchgängigen Entwicklung verteilter Systeme mit Echtzeitbedingungen für Rundrufnetze*. PhD thesis, University Karlsruhe, Faculty of Informatics, Institute for Program Structures and Data Organization (IPD), 2007. To appear.

[19] Marc Schanne and James J. Hunt. Remote Event Service Design. Technical report, FZI Forschungszentrum Informatik, 2004. Deliverable D4.2 describing the HIDOORS event channel network.

[20] Juha-Pekka Tolvanen. MetaEdit+: domain-specific modeling for full code generation demonstrated [GPCE]. In *Proceedings of the 19th Conference on Object Oriented Programming Systems Languages and Applications*, pages 39–40. ACM press, 2004.

[21] Andy J. Wellings et al. D1.2a. Technical report, University of York (on behalf of the HIJA project), 2005. HIJA deliverable D1.2a describing the analysis of requirements for high-integrity systems.

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