Evaluation of a Real-Time Distribution Service

Seppo Sierla
Jukka Peltola
Kari Koskinen
Helsinki University of Technology, Information and Computer Systems in Automation,
Konemiehentie 2, 02015 HUT
Tel. (09) 4515462, telefax (09) 4515394, Seppo.Sierla@hut.fi, Jukka.Peltola@hut.fi, Kari.O.Koskinen@hut.fi,
http://www.automationit.hut.fi

KEYWORDS distributed control system, distribution service, middleware, testing, RTPS

ABSTRACT

As control systems become more distributed and they are implemented with smaller hardware and software components, implementing the necessary communication links becomes challenging. There are considerable technical difficulties in guaranteeing upper bounds for latencies in motion or machine control, synchronizing the execution and communication of distributed components, taking corrective action in fault situations and reconfiguring the system.

In this paper, we discuss the main communication requirements of distributed control systems and use them to evaluate a certain distribution service product.

1 INTRODUCTION

The trends in automation have increased the importance of communication in automation systems. Systems controlling the factory need to be more transparent, letting real-time information about the machinery and production status flow through the plant up to the office level applications and even further to the supply network. Control functions are packaged as intelligent units aside process equipment offering their functionalities as manufacturing services to the higher level systems.

As the number of nodes in the system adds up to tens or over a hundred, the functional connections between the numerous nodes increase correspondingly. This makes implementing the various communication links more complex. In addition to traditional wired signals, the distribution service should also support various other communication mechanisms. The application designer should be able to work using only logical naming schemes without worrying about the locations of the corresponding data sources and consumers. Challenges in implementing the distribution services are e.g. latencies in motion control, synchronization of execution and communication of distributed applications, handling of faults and reconfiguring the system (on-line).

The paper discusses application level requirements for distribution services in automation systems and a distribution service product NDDS, which implements IDA-Group's RTPS standard (Interface for Distributed Automation, Real Time Publish-Subscribe). Some test results for NDDS are presented. The work has been done in connection with a Tekes project OHJAAVA-2 (2002-2003) /3,4/.

2 COMMUNICATION MECHANISMS

One task of the OHJAAVA project has been to extract and define the most essential communication mechanisms needed by and used in automation applications /5,11/. These logical level mechanisms carry out the tasks of communicating data and controlling execution of applications. An application designer uses these mechanisms when designing his automation applications, if they are supported by his engineering station and run-time environment. The most obvious mechanisms are *continuous data distribution*, *event-driven data distribution* and *event notification and acknowledgement services*.

Continuous data distribution is familiar to most automation engineers from the 'signal wiring' of various function block based automation languages like the IEC-61131-3 FBD. A wire between the ports of two function blocks models continuous updating of the target port with the value in the source port. The target port value is thought to be current, although the source might be located on another device node. A short cycle time

of program execution and data transfer gives an impression of truly continuous data flow and execution. Typical fast cycle times are 5-10ms, but many control tasks in process industries are satisfied with cycle times exceeding several minutes. Continuous data distribution has usually the strictest requirements for latency and jitter. The requirement for reliability, however, can be somewhat relaxed, because in continuous data distribution the next arriving sample can replace a lost message. The mechanism can be implemented either by transferring data cyclically or whenever the value has changed sufficiently. Especially in case of large data structures the latter is recommendable; however, this mode requires a reliable implementation where no samples are lost.

The **event-driven data distribution** mechanism is gradually finding its way to the application engineer's toolset (event-driven, IEC 61499-1). In this mechanism, data is transferred to its destination only as the state of the producing block or component changes, i.e. when an event occurs. The simplest event message only carries the name or type of the event. More realistically, however, the message carries various payload data, timing and quality information and other attributes. In event-driven data distribution, it is essential that messages reach all registered destinations. The information must be timely and without conflict with data in other nodes of the distributed system. In addition to just transferring data, an event may trigger the execution of an algorithm in the receiving component and node. This may cause further state changes that create new events. In this way, the event-driven data distribution mechanism can handle the execution control of a distributed application. The execution is hereby synchronized by a local or remote event instead of a cyclically scheduled operating system task.

The **event notification and acknowledgement** mechanism is used in automation systems mainly for delivering alarms and notifications (later just notifications). Notifications are usually first shown in some user interface and then logged in a database. An essential part of the mechanism is that the notification is acknowledged at the user level after it has been received. Another feature, which makes this different from the previous event-driven data distribution mechanism is that the receiving party usually orders large groups of different notification types from various sources, instead of having a 'wired connection' to a single component or a channel. 'Wildcards' and hierarchical naming structures are used to make ordering simple. The event notification and acknowledgement mechanism also requires a reliable implementation mechanism.

3 REAL-TIME PUBLISH-SUBSCRIBE

IDA's (Interface for Distributed Automation) /1/ RTPS (Real-Time Publish Subscribe) communication standard is one promising middleware solution. RTI's /8/ NDDS implementation is the only commercially available product; the next version of NDDS that also implements OMG's recent Data Distribution Service specification will be available later this year /8/. RTPS is only a communication protocol, so the implementers of NDDS had much freedom in designing the architecture /12/. This article concentrates on the communication mechanisms.

3.1 Implementation-Level Communication Mechanisms

NDDS supports continuous data distribution, event notifications and request-replies /6,9,11/. With every mechanism, the programmer is given an opportunity to set parameters that control the Quality of Service (QoS). In this way, reliability, memory usage and determinism can be optimized according to the needs of the application.

3.1.1 Continuous Data Distribution

Figure 1 illustrates the publish-subscribe mechanism. For continuous data distribution, the 'best-effort' version of this mechanism is most appropriate. It has been designed for minimal latency, so messages that are lost by the network are not retransmitted. This is usually the best approach for transmitting periodic or frequently sent measurement and control signals.

The flexibility of the publish-subscribe mechanism derives from the fact that publications and subscriptions do not need to know anything about each other. A signal is identified by a topic name, such as 'temp' or 'pressure' in the figure. A producer of measurement data needs to create a publication for a topic, and after this individual measurement values can be sent as issues by making one method call. A consumer of data can create a subscription as long as it knows the name of the topic it is interested in. The middleware will then interrupt the consumer application every time it receives new data. Therefore, producers of data do not need to know anything about the number and location of possible consumers, because the middleware maintains the necessary databases. Reconfiguring and updating the application becomes easier, and there is little room for error.

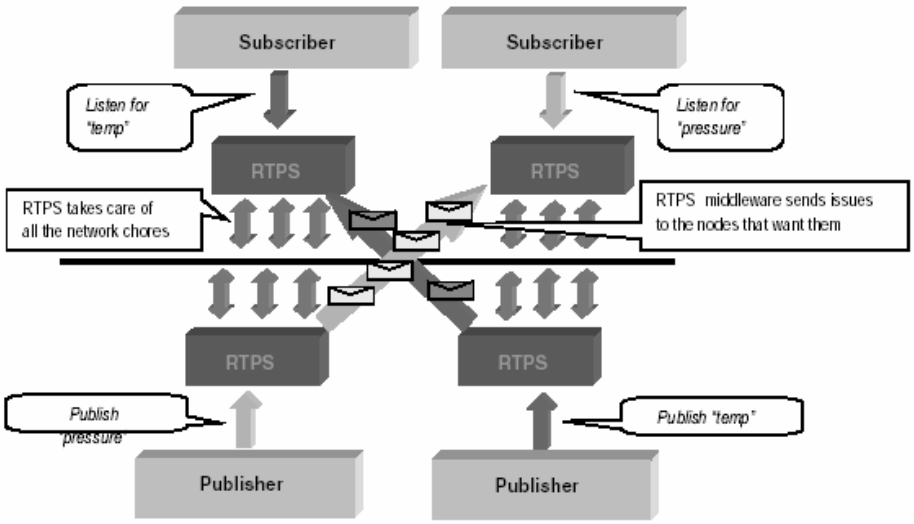

Figure 1 The publish-subscribe mechanism /9/

3.1.2 Request-Reply

The request-reply mechanism of RTPS resembles the remote procedure call that is widely used in object-oriented programming. The middleware makes services available by name (clients do not need to know the location of the server), and there is support for QoS and redundant servers.

3.1.3 Event Notification and Acknowledgement Services

The reliable version of publish-subscribe is well suited for transmitting event messages. The implementation uses retransmissions and acknowledgements very much like TCP, but better QoS support is provided. These messages might belong to a sequence of commands, so they must arrive in-order and none may be lost. However, alarms can also be transmitted, and the pattern subscription's of NDDS enable a programmer to subscribe to all topics whose name fields satisfy the specified filter criteria. Reliable publish-subscribe also satisfies our requirements for event-driven data distribution reasonable well. However, there is no support for operator-level acknowledgements of events and alarms.

4 TEST ARRANGEMENTS

Our goal was to test the behavior of the middleware products' communication mechanisms. Therefore, no process was simulated, but the test components generated traffic whose quality and quantity resembled the communication of a small automation system. Continuous data distribution, event notifications and scalability were tested. In this article, we can only present the basic scenarios for continuous data distribution and system scalability; all results are fully documented in /10/.

The following test components were implemented above the middleware-layer using NDDS's API. Test cases are defined by assigning components to nodes and specifying the parameters of each component.

Cyclic signal generator: The generator creates a publication for each topic name that is given to it as a parameter. With a period P, a new measurement value is generated for each topic and issues are sent. Since no process is simulated, only a double timestamp and integer sequence number are sent. This and the next component use the best-effort publish-subscribe mode.

Cyclic signal receiver: The receiver creates a subscription for each topic that it gets as a parameter. The subscriptions are also given a deadline, so that NDDS will notify the application if new data is not received before the deadline expires. The component measures the latencies for each issue, and after the test run is over, the minimum, maximum, average and standard deviations of these are computed. Any missing sequence numbers are also recorded.

Event-based communication was tested with **event generator** and **receiver** components, but they are not involved in the tests that are presented here.

The tests were run on the following nodes (the last part of the IP address is used as the name of the node.) The tests in section 5.1 used only the first three nodes in the list.

45:	(667MHz, 256MB RAM, Windows XP)
72:	(400MHz, 128MB RAM, Mandrake 8.2 Linux)
199:	(2GHz, 512MB RAM, Windows XP)
89:	(533MHz, 128MB RAM, Mandrake 8.2 Linux)
127:	(350MHz, 128MB RAM, Mandrake 8.2 Linux)
92:	(533MHz, 64MB RAM, Mandrake 8.2 Linux)

The nodes were connected with a 100Mbps switched Ethernet.

5 TEST RESULTS

5.1 Continuous Data Distribution

The test case for the basic scenario is presented here, where one publication sends data to three subscriptions. The signal generator that creates the publication is on node 199 and it sends 10000 issues with a period of 20ms. Signal receivers that subscribe to this topic are started on nodes 199, 45 and 72. The deadline for all subscriptions is 40ms.

The receivers measured the latencies for each issue and calculated the minimum, maximum, mean and standard deviation after the test was over. Figure 2 shows these statistics for each node. However, the maximums are omitted, because they would not have fit into this figure. They were 0.57ms, 5.5ms and 7.2ms for nodes 45, 72 and 199, respectively.

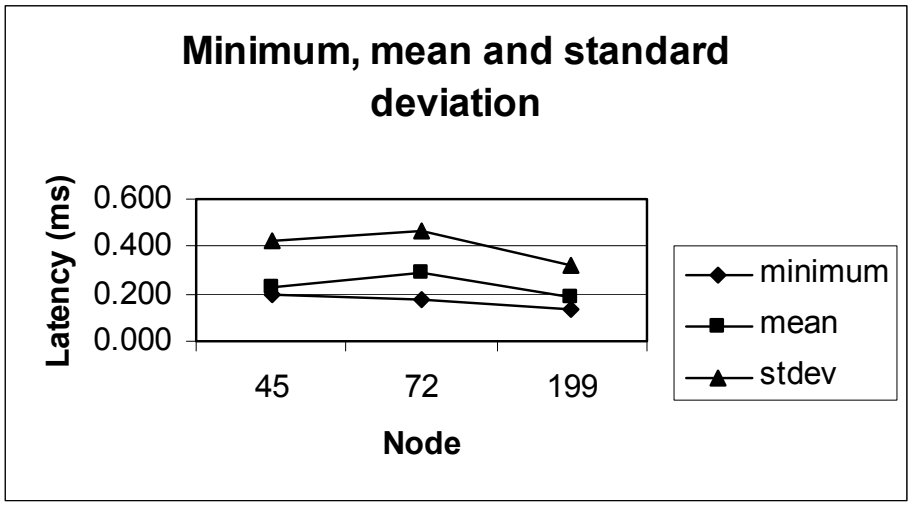

Figure 2 Comparison of three receivers

It must be remembered that the issues that were sent to nodes 45 and 72 went over a 100Mbps Ethernet. The processor on 199 was 3-4 times faster than the other ones, but a generator and receiver component were competing for its time. The results for 45 and 72 are quite close to those for 199, so the network's contribution to latencies was relatively small. Most latencies were quite close to the mean, but there were a few latencies of several ms that had a strong effect on the standard deviation. The operating system's scheduler is presumably responsible for this behavior, because deterministic results can only be obtained if the middleware's sending and receiving threads are scheduled in a timely fashion. Although the components were run on high priority, Windows XP and Mandrake Linux do not provide any guarantees. NDDS has been designed to process all outgoing and incoming issues as fast as possible as they come /12/. Since the sending rate was easily handled by these CPUs, we consider the scheduling to be the main cause for indeterminism in this test.

All of the test components started their initialization routines when a startup signal was broadcasted, so the publication started to send issues as soon as the signal generator's setup calls had returned. The receivers started to get data after the subscriptions were ready and the NDDS middleware had established the communication links between publishing and subscribing applications. Since we did not use any wait calls that return when

publications and subscriptions have detected each other, some of the first issues were lost. In this test, all receivers got issues starting from number 30, and none were missed after this. The first issue was received between 599 and 608ms after the receiver component got the startup signal, which is the time it takes to send 30 issues at a 20ms period. The publication must have started within a few ms, and the 'wiring' between the components was established in ca. 600ms. After this, the behavior was as we expected, and the 40ms deadline did not cause any alerts.

5.2 The Scalability Test

Figure 3 illustrates the basic scenario of the scalability test, where all of the publications and subscriptions are engaged in cyclic data transfer. A topic name next to a node indicates that the signal generator on that node creates 100 publications using that name as a prefix. The adjacent time is the period with which new issues are sent for each publication. For example, the generator on node 199 creates publications for topics temp1, temp2, … , temp100 and sends an issue for each of these every 20ms, resulting in 5000 issues per second. An arrow from node A to node B means that B has a receiver that creates subscriptions for every topic that node A publishes. The receiver on 89 has 200 subscriptions, one for each pressure and current topic.

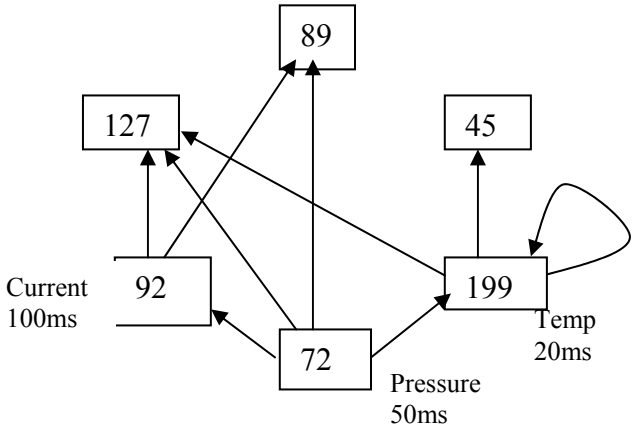

Figure 3 The configuration of the scalability test

Much data was obtained even from the basic scalability test, so here we will examine the results for the subscriptions on node 199. The 100 pressure topics followed by the 100 temp topics are on the horizontal axis of Figure 4. This has curves for the minimum, mean, maximum and standard deviation, and each topic contributes one point onto each curve. For example, the maximum of 'temp39' is 4.6ms, as can be seen by reading the value of the max curve at 139. The figure should be discrete, but with 200 points per curve it is much clearer to use continuous lines.

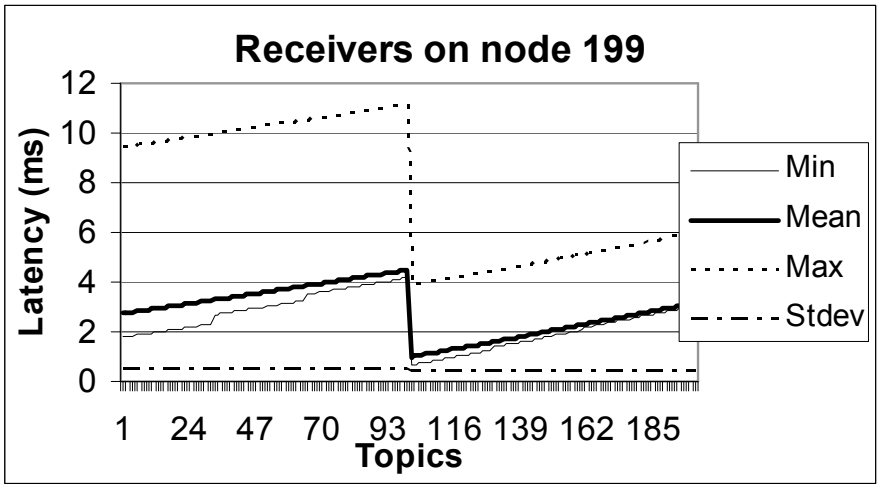

Figure 4 Statistics for subscriptions on 199

For each group of topics that is published by one generator, we can see that the means increase linearly. RTPS does not specify the order in which issues are sent (since the publications were grouped under one publisher group-object), but in this case the order clearly was 'temp1', 'temp2', 'temp2', … and they have been received in this order. The means are greater for topics that were sent later, because the CPU on the receiving node is a bottleneck. (We verified this by adding some processing to the receiver's issue handling routine, and the steepness of the curves increased.) The minimum and maximum curves behave similarly and the standard deviation varies little between topics. Since the curves are so regular, we can say that the performance is reasonably predictable, although hard guarantees cannot be made on these operating systems. These results give a good indication to the best, average and worst case performance under certain circumstances; many other typical scenarios are covered in /10/. Whether any certain topic gets the best or worst case performance depends ultimately on the internal workings of the middleware. RTPS and NDDS do not offer any policy to prioritize the traffic of some topics, since any such processing would decrease the overall performance /12/.

6 CONCLUSIONS

We consider RTPS to be an interesting middleware solution for process automation applications. Our results suggest that the performance of the NDDS implementation is sufficient for modern automation systems, and that the reliability of the data transfer is satisfactory even under peak load. However, our test runs were relatively short (typically 10000 issues per publication) and the configuration of the scalability test only corresponds to a small automation system. The performance could have been more deterministic, but this can be amended by using a RTOS. Over the course of our testing, we had to reconfigure the system frequently, but the logical application model of RTPS made this work easy. The distribution service also noticed run-time configuration changes quickly (usually it took under a second to establish data-flows.) We tested porting the components to Linux by editing the code for the Windows versions, and no changes were required to any of the calls to the NDDS API.

No available middleware product satisfies all of our requirements, but NDDS seems promising. Real-time behavior and determinism could hardly be significantly better on these platforms, although performance has been optimized at the cost of determinism. NDDS has been used, for example, on the VxWorks platform in applications, whose real-time requirements exceed those in typical automation applications /8/. Our experiences with static and dynamic reconfiguration, using the NDDS API, porting the code to different operating systems and developing applications quickly were clearly positive. However, since NDDS has not been designed specifically for the process automation industry, it has some shortcomings in guaranteeing the integrity of signal groups as well as handling alarms and operator level acknowledgements. None of these difficulties are insurmountable /10/.

7 BIBLIOGRAPHY

1. Interface for Distributed Automation, http://www.ida-group.org/, referred 8.8.2003
2. IDA: White Paper V.1.0 April 2001, http://www.ida-group.org/ referred 8.8.2003
3. OHJAAVA-2 project's homepage at HUT http://www.automationit.hut.fi/tutkimus/documents/Ohjaava/eohjaava-2.htm, referred 15.4.2003
4. OHJAAVA project group's homepage at VTT http://www.vtt.fi/tuo/projektit/ohjaava/, referred 15.4.2003
5. Tommila T.: Automaation uudet hajautusratkaisut ja hajautuksen vaatimukset, OHJAAVA-2 projektin työdokumentti
6. Sierla S.: Analysis of Middleware Solutions – RTPS, OHJAAVA-2 project's technical document, unpublished, 10.2.2003
7. Peltola J.: Automaatiojärjestelmien hajautusta palvelevat arkkitehtuurit, ÄLY-foorumi, Seinäjoki, 15.5.2003
8. Real Time Innovations Inc., http://www.rti.com, referred 10.05.2003
9. RTI: NDDS User's Manual Version 3.0. February 2002.
10. Sierla S.: Middleware Solutions for Automation Applications – Case RTPS, Report 9, Laboratory for Information and Computer Systems in Automation, HUT, June 2003
11. Tommila T. et al.: Uudet hajautusratkaisut avoimissa automaatiojärjestelmissä, Automaatio 03

Interviews and conversations:

12. Howard Wang, RTI, 9.2002 – 6.2003