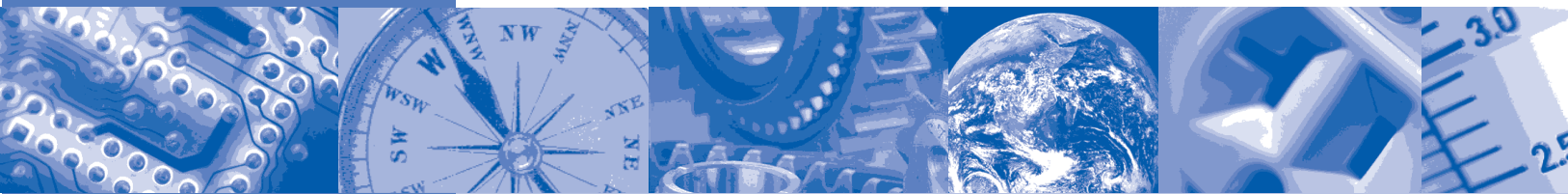


ISA-TR50.02, Parts 3 & 4-2000



Fieldbus Standard for Use in Industrial Control Systems, Parts 3 & 4: Technical Report for Fieldbus Data Link Layer — Tutorial



ISA—The Instrumentation,
Systems, and
Automation Society

Approved 15 April 2000



Documents

Document Sharing Hub

ISA-TR50.02, Parts 3 & 4-2000
Fieldbus Standard for Use in Industrial Control Systems, Parts 3 & 4:
Technical Report for Fieldbus Data Link Layer — Tutorial

ISBN: 1-55617-733-X

Copyright © 2000 by the Instrument Society of America. All rights reserved. Not for resale. Printed in the United States of America. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means (electronic mechanical, photocopying, recording, or otherwise), without the prior written permission of the Publisher.

ISA
67 Alexander Drive
P. O. Box 12277
Research Triangle Park, North Carolina 27709

Preface

This preface, as well as all footnotes and annexes, is included for information purposes and is not part of ISA-TR50.02, Parts 3 & 4-2000.

This document has been prepared as part of the service of ISA, the international society for measurement and control, toward a goal of uniformity in the field of instrumentation. To be of real value, this document should not be static but should be subject to periodic review. Toward this end, the Society welcomes all comments and criticisms and asks that they be addressed to the Secretary, Standards and Practices Board; ISA; 67 Alexander Drive; P. O. Box 12277; Research Triangle Park, NC 27709; Telephone (919) 549-8411; Fax (919) 549-8288; E-mail: standards@isa.org.

The ISA Standards and Practices Department is aware of the growing need for attention to the metric system of units in general, and the International System of Units (SI) in particular, in the preparation of instrumentation standards. The Department is further aware of the benefits to USA users of ISA standards of incorporating suitable references to the SI (and the metric system) in their business and professional dealings with other countries. Toward this end, this Department will endeavor to introduce SI-acceptable metric units in all new and revised standards, recommended practices, and technical reports to the greatest extent possible. *Standard for Use of the International System of Units (SI): The Modern Metric System*, published by the American Society for Testing & Materials as IEEE/ASTM SI 10-97, and future revisions, will be the reference guide for definitions, symbols, abbreviations, and conversion factors.

It is the policy of ISA to encourage and welcome the participation of all concerned individuals and interests in the development of ISA standards, recommended practices, and technical reports. Participation in the ISA standards-making process by an individual in no way constitutes endorsement by the employer of that individual, of ISA, or of any of the standards, recommended practices, and technical reports that ISA develops.

CAUTION — ISA ADHERES TO THE POLICY OF THE AMERICAN NATIONAL STANDARDS INSTITUTE WITH REGARD TO PATENTS. IF ISA IS INFORMED OF AN EXISTING PATENT THAT IS REQUIRED FOR USE OF THE STANDARD, IT WILL REQUIRE THE OWNER OF THE PATENT TO EITHER GRANT A ROYALTY-FREE LICENSE FOR USE OF THE PATENT BY USERS COMPLYING WITH THE STANDARD OR A LICENSE ON REASONABLE TERMS AND CONDITIONS THAT ARE FREE FROM UNFAIR DISCRIMINATION.

EVEN IF ISA IS UNAWARE OF ANY PATENT COVERING THIS STANDARD, THE USER IS CAUTIONED THAT IMPLEMENTATION OF THE STANDARD MAY REQUIRE USE OF TECHNIQUES, PROCESSES, OR MATERIALS COVERED BY PATENT RIGHTS. ISA TAKES NO POSITION ON THE EXISTENCE OR VALIDITY OF ANY PATENT RIGHTS THAT MAY BE INVOLVED IN IMPLEMENTING THE STANDARD. ISA IS NOT RESPONSIBLE FOR IDENTIFYING ALL PATENTS THAT MAY REQUIRE A LICENSE BEFORE IMPLEMENTATION OF THE STANDARD OR FOR INVESTIGATING THE VALIDITY OR SCOPE OF ANY PATENTS BROUGHT TO ITS ATTENTION. THE USER SHOULD CAREFULLY INVESTIGATE RELEVANT PATENTS BEFORE USING THE STANDARD FOR THE USER'S INTENDED APPLICATION.

HOWEVER, ISA ASKS THAT ANYONE REVIEWING THIS STANDARD WHO IS AWARE OF ANY PATENTS THAT MAY IMPACT IMPLEMENTATION OF THE STANDARD NOTIFY THE ISA STANDARDS AND PRACTICES DEPARTMENT OF THE PATENT AND ITS OWNER.

ADDITIONALLY, THE USE OF THIS STANDARD MAY INVOLVE HAZARDOUS MATERIALS, OPERATIONS OR EQUIPMENT. THE STANDARD CANNOT ANTICIPATE ALL POSSIBLE APPLICATIONS OR ADDRESS ALL POSSIBLE SAFETY ISSUES ASSOCIATED WITH USE IN

HAZARDOUS CONDITIONS. THE USER OF THIS STANDARD MUST EXERCISE SOUND PROFESSIONAL JUDGMENT CONCERNING ITS USE AND APPLICABILITY UNDER THE USER'S PARTICULAR CIRCUMSTANCES. THE USER MUST ALSO CONSIDER THE APPLICABILITY OF ANY GOVERNMENTAL REGULATORY LIMITATIONS AND ESTABLISHED SAFETY AND HEALTH PRACTICES BEFORE IMPLEMENTING THIS STANDARD.

ISA would like to thank the Italian Electrotechnical Commission for their contributions to this document.

The following members of ISA Subcommittee SP50.3 contributed to the development of this document:

NAME	COMPANY
A. Gupta, Chair	Globespan Inc.
H. Dammeyer, Managing Director	The Ohio State University
G. Wood, Managing Director	Graeme Wood Consulting
P. Andreassen	Fieldnet As
W. Borst	Borst Automation
A. Capel	Comgate Engineering Limited
S. Cavalieri*	Universita Deglistudi di Catania
R. Crowder	Ship Star Associates Inc.
L. De Souza	SMAR Equipamentos Ind Ltda
B. Dumortier	Schneider Electric SA
C. Gross	Dow Chemical Company
D. Hewitson*	Ronan Engineering Company
T. Ito	Fuji Electric
V. Jacobson*	Ronan Engineering Company
J. Lerare	Cegelec
P. Leviti	Marconi Italiana Automazione
K. Lindner	Endress + Hauser Process Solutions AG
O. Mirabella*	Universita Degli Studi di Catania
L. Oreans	Siemens AG
S. Pedersen	ABB Corporate Research
T. Phinney	Honeywell Inc.
D. Serang	MX Communications
K. Shin	University of Michigan

The following members of ISA Committee SP50 contributed to the development of this document:

NAME	COMPANY
R. Caro, Chairman	ARC Advisory Group
H. Dammeyer, Managing Director	The Ohio State University
G. Wood, Managing Director	Graeme Wood Consulting
C. Akiyama	Yokogawa Electric Corporation
J. Aliperti	Chromatech Consultoria Limited
W. Baker	KML, Inc.
L. Beckman	HIMA Americas, Inc.
M. Borges	Petrobras Cenpes Supen Diprom
R. Bowden*	Fisher-Rosemount Ltd.
S. Boyer	Iliad Engineering Inc.

* One vote per company.

R. Breithaupt*
P. Brett*
P. Burton
R. Canfield
A. Capel
B. Casner
S. Cavalieri*
R. Crowder
E. dela Hostria
U. Doebrich*
J. Dohoney
B. Dumortier
J. Elias*
T. Feindel
M. Giraudi
J. Gray
P. Griem, Jr.*
B. Gross
C. Gross*
A. Gupta
H. Hasler
T. Ito*
V. Jacobson*
J. Jamison
K. Kakizakai*
R. Ketcham
K. Krivoshein*
Y. Kumeda
T. Kuusisto
T. Laine*
H. Landet
C. Langford
R. Lasher*
P. Leviti
A. Lesh
K. Lindner
T. Madden*
G. Mazzola*
A. McCauley, Jr.
F. McKenna
R. Meeker
D. Merriman
O. Mirabella*
D. Modell
A. Mukherji
G. Nachev*
L. Natiello
L. Neitzel*
H. Newman
P. Noury*
E. Pageler*

Siemens Schweiz AG
Honeywell, Inc.
Holwell Consulting
Square D Company
Comgate Engineering, Ltd.
Miller Electric Manufacturing Company
Universita Degli Studi di Catania
Ship Star Associates, Inc.
Rockwell Automation
Siemens AG
Jet Propulsion Laboratory
Schneider Electric SA
Honeywell, Inc.
R. Stahl, Inc.
Consultant
Foxboro Company
Honeywell Inc.
General Motors Corporation
Dow Chemical Company
Globespan Inc.
Adelcontrol
Fuji Electric
Ronan Engineering Company
Bantrel Inc.
Fuji Electric Company, Ltd.
Union Carbide Corporation
Fisher-Rosemount Systems Inc.
Yamatake Industrial Systems
Consultant
Alstom Technology
FINT
Cullen G. Langford Inc.
Consultant
Marconi Italiana Automazione
AMP Inc.
Endress + Hauser Process Solutions AG
Exxon Company USA
ENEL
Chagrin Valley Controls, Inc.
Catena Associates
Process Controls Solutions, Inc.
MerTech, Inc.
Universita Degli Studi di Catania
ABB (Bailey Controls)
Systems Automation International Inc.
Isomatic Laboratories
Merck and Company Inc.
Fisher-Rosemount Systems Inc.
Cornell University
Alstom
Fisher-Rosemount Systems Inc.

* One vote per company.

P. Patel
 T. Phinney*
 A. Poschmann
 F. Russo*
 K. Schwarz
 E. Sederlund*
 Y. Shimanuki
 E. Skabowski
 J. Slavinsky
 D. Snizek
 J. Sprague
 R. Squires
 H. Storey
 R. Szanyi*
 P. Thiessmeier
 N. Tobol*
 A. Tresset*
 C. Vaidya
 C. Van Haren
 J. Weiss*
 C. Williams
 G. Winchester
 D. Wroblewski*
 M. Zielinski*

V Automation
 Honeywell Inc.
 IFAK
 ENEL-Ricerca/PEA
 Schwarz Consulting Company
 Dow Chemical Company
 Toshiba Corporation
 Tendent Consulting
 Czech Standards Institute
 Lockheed Martin Federal Services
 Saudi Aramco Oil Company
 Measurement Technology Ltd.
 Equilon Enterprises LLC
 Exxon/Mobil Research Engineering
 Klockner-Moeller
 Ronan Engineering
 Cegelec
 Sun Microsystems
 James River Corporation
 Electric Power Research Institute
 Eastman Kodak Company
 National Electric. Manufacturers Association
 Bailey Controls Company
 Fisher-Rosemount Systems Inc.

This Standard was approved for publication by the ISA Standards and Practices Board on 15 April 2000.

NAME

COMPANY

M. Zielinski
 D. Bishop
 P. Brett
 M. Cohen
 M. Coppler
 B. Dumortier
 W. Holland
 A. Iverson
 R. Jones
 V. Maggioli
 T. McAviney
 A. McCauley, Jr.
 G. McFarland
 R. Reimer
 J. Rennie
 H. Sasajima
 R. Webb
 W. Weidman
 J. Weiss
 J. Whetstone
 M. Widmeyer
 R. Wiegler
 C. Williams
 G. Wood

Fisher-Rosemount Systems, Inc.
 Chevron Petroleum Technology Co.
 Honeywell, Inc.
 Senior Flexonics, Inc.
 Ametek, Inc.
 Schneider Electric SA
 Southern Company
 Ivy Optiks
 Dow Chemical Co.
 Feltronics Corp.
 Instrumentation & Control Engineering LLC
 Chagrin Valley Controls, Inc.
 Honeywell, Inc.
 Rockwell Automation
 Factory Mutual Research Corp.
 Advanced Architecture and Technologies
 Altran Corp.
 Parsons Energy & Chemicals Group
 EPRI
 National Institute of Standards & Technology
 EG&G
 CANUS Corp.
 Eastman Kodak Co.
 Graeme Wood Consulting

* One vote per company.

Contents

Introduction	9
1 Incomprehensibility	9
1.1 Standard style.....	9
1.2 Features variety.....	9
1.3 Many updates	9
1.4 Effort to understand	9
2 Tutorial target.....	9
Foreword	11
Part A	13
A1 Scheduled access vs. circulated token	13
A2 Buffers vs. queues	33
A3 Connections vs. connectionless.....	37
A4 Addressing	50
A5 Traffic types.....	53
A7 Timeliness	62
A8 Starting a node.....	67
A9 Bridges	68
A10 Segmentation	72
A11 Simple devices support.....	72
Part B	74
B1 DLL simplified overview	74
B2 DLL scheduling diagrams.....	85
B3 Slot-time	87
B5 Extension of PIII-table 10 (not available yet)	89
B6 How to build EC parameters	89

B7a	DLC establishment diagrams.....	92
B7b	DLC reset diagrams	102
B8	DLC establishment examples (not yet available).....	112
B9	DLC data traffic mechanisms.....	112
B10	Reset purging logic mechanisms (not yet available)	119
B11	Addressing tables.....	119
B12	Addressing examples.....	122
B13	Traffic slides (not yet available)	123
B14	LM+LAS traffic (not yet available).....	123
B15	Time distribution mechanisms (not yet available).....	123
B16	How simple a device can be (not yet available).....	123
B17	Timeliness extract (not yet available).....	123

Introduction

1 Incomprehensibility

A long story of incomprehensibility has followed (and often also preceded) the ISA Fieldbus Data Link Layer (DLL) documents. That is due to several reasons: some real ones, others only imagined but repetitively suggested and spread around by some of the people who did not agree with the proposal.

1.1 Standard style

First of all, the fear of compelling something relevant only to the implementation domain always pushes standard documents not to describe "why" or "how" but just "what" to do.

That, in this particular case of fieldbus, made things even worse than usual because of the existing experiences and approaches of many experts being radically different; it led to real problems (and it eased many imagined ones) in mutual understanding.

1.2 Features variety

Due to the differences between the existing fieldbus approaches, the technical and diplomatic need of merging them has always been the key issue of all the ISA DLL efforts and has so increased the variety of features and the complexity of the structure within such a document that tries to harmonize them all.

1.3 Many updates

Then, the long path followed in order to chisel a proposal that could be really agreed by all the different existing approaches has caused the need to frequently update, partially re-write, or delete, or add many parts of the ISA Fieldbus DLL drafts along several years.

And this, of course, did not give the documents the same clear structure as an immediate paper would have had.

1.4 Effort to understand

Finally, we have to remind how only "if" and "when" we really want to implement something, then we make the effort to understand it. When we already have an existing, even if not universal, alternative why should we make such an effort?

2 Tutorial target

Well, as done in previous editions, this tutorial aims to fill such need of explaining the "why" and "how" relevant to "what" has been included in the ISA Fieldbus DLL documents.

Differently from the past, when the addressed audience was mainly made of the members themselves of the ISA committees (see last part of 1.1 above), this edition would also like to address the people who, so far, have not had any contact with the ISA Fieldbus DLL documents. At the same time, we would like to save all the detailed tutorial material prepared in the past. In the foreword below we'll see how this kind of compromise has been reached.

This page intentionally left blank.

Foreword

We have chosen to split the tutorial in two parts:

Part A - to ease the first approach to the ISA Fieldbus DLL documents by means of the following goals:

- Presenting the peculiar reasons why some of the features have been included
- Giving the flavor of its main features
- Making simple examples or graphic representation of such features

Part B - to ease the eventual consequent study of the ISA Fieldbus DLL documents by means of the following goals:

- Linking separate parts of this standard relevant either to the same subject or to correlated items
- Making detailed examples of this DLL mechanism

Some of the clauses of Part B are not yet available; that is also because Part B is a collection of already existing works done during the ISA committee activity. Still we decided to publish this tutorial even if Part B is incomplete, both for the reasons listed in its Introduction, which seem to urge its Part A and as an invitation too for some other committee members to work on it in some future joint efforts.

Whenever possible, all parts and subparts of the tutorial refer to the relevant ISA Fieldbus DLL documents clauses and subclauses to help in tracing each subject all along the standard documents.

The **following planning table**, for each one of the above parts and relevant goals, relates the DLL main features, which are described in this tutorial, to the relevant clauses/subclauses of this document.

NOTE — By using such a table, the reader, only interested to an external viewpoint over this DLL, could go rapidly through the "reasons why features exist," "examples/graphic representations" and the "flavor" of just those corner-stone features marked by "*".

As a general rule, we will use prefixes "A" and "B" to indicate a clause/subclause contained in the two parts of this document while PIII and PIV will refer to clauses/subclauses of, respectively, Part III (DLL Services) and Part IV (DLL Protocol) of the ISA Fieldbus Document (to be more precise, of ISA-S50.02, Part 3-1997 and ISA-S50.02, Part 4-1997).

Tutorial Planning Table

Tutorial parts and goals:	Part A			Part B	
	reasons why features exist	flavor of features	examples/ graphic represent.	linking of separate parts	detailed examples & graphic represent.
DLL main features:					
*Sche. access Vs. Circ. Token	t: A1.1.x	t: A1.2	G: fig.1-3		G: B3
Scheduled access					
LAS		t: A1.2.1.x			G: B14
Sequences		t: A1.2.2		T: B4	G: B2
Circulated Token					
Token-time policy		t: A1.2.3			
Repetitive sequences	t: A1.2.4.1	t: A1.2.4.2			
*Buffers Vs. Queues	t: A2, A2.1	t: A2.2	E: fig.4.x		G: B2
*Connections Vs. Connectionless	t: A3.1	t: A3			
Connection-oriented					
Conn. types & features	t: A3.2	t: A3.2.1		t: B9, T: B5	G: B10
Connection establishment		t: A3.2.2		t: B6, T: B4	G: B7, E: B8
Connectionless					
Acknowledged	t: A3.3.1	t: A3.3.2		T: B4	
Two ways exchange	t: A3.4.1	t: A3.4.2			
*Addressing		t: A4		T: B4	T: B11, E: B12
Traffic types					G: B13
Data traffic			G: fig. 5		
Non-data traffic					G: B14
Relat.ship services/traffic				G: B1	
Time distribution & Synchroniz.	t: A1.2	t: A6		t: B15	
*Timeliness	t: A7.1	t: A7.2	G: fig. 7	t: B17	
Starting a node		t: A8			
Bridges	t: A9		G: fig. 8		
Segmentation	t: A10				
Simple devices support (+pre-establish. & grouping)	t: A11	t: A11.1/2		t: B16	

Legend: t=text, T=table, G=graphic representation, E=example

Part A

A1 Scheduled access vs. circulated token

A1.1 The long war

It's easy to recall in this issue of tokens and scheduling the main subject of so many battles between fieldbus experts coming from different experiences. Each one of them has tried (and some of them still try) to impose his own view: the only good solution versus a partial, inappropriate, faulty alternative!

And now the ISA Fieldbus DLL documents come saying, **"Both paradigms, circulated token and scheduled access were good, but insufficient at the same time; they were complementary, not alternative, and a complete fieldbus solution needs the two together."**

How is this possible?

We'll try to show that need by using a silly (but suitable, we believe) analogy.

A1.1.1 Cars vs. trains analogy

When the first cars appeared at the end of the nineteenth century, after a short dismay it was easy to foresee the car itself as "the solution" of any man's problem regarding short-medium range transportation.

Of course people realized that some adaptations were needed by the existing roads (asphalt, signaling, widening), but at that time nobody thought a car transportation system would have ever collapsed.

The problem was (and always is) that once the means to satisfy a need is provided then the need's size enlarges beyond any possible previous expectation.

So it happened with cars. Soon they were used not just to satisfy the previously existing needs of transportation, but they generated new requests by themselves. The fact they existed gave people new ideas about what, when, where to transport! Now it was so easy, why not to use a car for it?!

There the collapse originates from; no matter how many, large, modern facilities we build, people's transportation needs always rise (really pushed by the facilities themselves) to a point at which they saturate. It's just hopeless because it's automatic; the freedom of access to such means brings its potential demand out of any possible control. At the topic moment even the ambulance will have to follow the slow procession within the traffic jam.

That's why in the last years we have seen a strong recovery of trains (or train-like means) on short-medium range transportation; they fulfill the need of commitment on a pre-established time-table (schedule).

Do we need to be at work in the city center at eight o'clock every morning, regardless of weather or traffic conditions? Do we need to reach Florence in two hours from Rome, regardless of being on a summer holiday weekend?

Let's plan a train journey then.

But does this mean that we have to forget cars?

Certainly not; trains are not the correct or best answer to many other not-scheduled transportation needs: would we like to wait for a train when we wish to pop over for visiting some friends, or to pay for a railway that could reach even our personally preferred pub?

So it's just obvious, we need a good mix of cars and trains, so well balanced as to not get the trains when they aren't really needed, but always giving priority to trains over cars at each crossing.

A1.1.2 Fieldbus case

The same happens with fieldbus where the objective difference is just to transfer data frames over an industrial site instead of transporting people over a geographical area.

When the first fieldbuses were proposed, the existing needs of data traffic on industrial plants could have been satisfied by nearly any proposed solution. But such a nice and flexible means soon stimulates the potential data traffic and networks extension beyond any possible previous expectation. Now it's so easy, why not add an extra data on fieldbus for making such a feature configurable? Now it's so easy, why shouldn't we connect that equipment to our network too?

There the collapse of fieldbus free access mechanism originates, no matter how fast, carefully-constrained fieldbus facilities we build, data transportation needs always rise (really pushed by the facilities themselves) to a point at which they saturate. It's just hopeless because it's automatic; the freedom of access to such a means brings its potential demand out of any possible control. At the topic moment even the most urgent data may have to wait for the token to come back to it, and if the connected equipments are not a few, if each one has something to transfer, if . . . , then the worst case of the response to such an urgent need cannot be kept as small as we would like.

That's where a scheduled access can instead fulfill the commitment. Do we need to update information every 10 msec, regardless of traffic conditions? Do we need to know about an event within 20 msec of it happening, regardless of how much equipment is connected and of how much they have to transfer?

Let's plan a scheduled access then.

But does this mean that we have to forget tokenized access?

Certainly not; scheduled access is not the correct or best answer to many other data transfer needs. Would we like to wait for a pre-scheduled access when we already have data to transfer and nobody else is using the network, or to pay bandwidth for a periodic scheduled access when we have noncritical and/or episodic data to be transferred just once?

So, even if it doesn't seem to have been obvious, we need a good mix of circulated token and scheduled access, so well balanced as to not have to pay bandwidth for the scheduled access when it isn't really needed, but always giving priority to scheduled access over circulated token when a conflict arises.

A1.1.3 ISA philosophy

That's what the ISA Fieldbus DLL documents propose: an overall schedule able to guarantee the needed data at the needed time but also allowing gaps within which a circulated token mechanism can take place while complying with a defined maximum rotation time.

Going back to the previous analogy, ISA Fieldbus DLL documents propose a combined system of roads and railways where, at each level-crossing, cars stop to let trains cope with their time table, but where it is also possible to define the limit of the number of trains in order to guarantee a maximum waiting length for the cars at the barrier.

A1.2 ISA solution

Such a philosophy is clearly a mix of authority and freedom, of centralized government and federal autonomy; its implementation cannot avoid nominating a kind of democratic "king" who univocally imposes the transmission of a defined data at a defined time by a defined people, when so required, but also guarantees a defined minimum amount of free time to each one of its people for their chats. In fact

- on one side, a fair method to guarantee a pre-established transfer of a defined data at a defined time is to have one and only one "decision maker," which compels all the others' transmissions preventing them from arguing about who's going to do what (see figure A1); and
- on the other side, a fair method to give everyone equal possibility to freely access the fieldbus is to let a token pass among all the nodes in a circular repetitive way: each node receives such token from the "preceding" node, uses it for its needs up to a limited amount of time and then passes the token itself to the "successive" node (see figure A2).

NOTE — We take the liberty of using "node" instead of the more appropriate and precise "Data Link Entity" (DLE) because we think it helps a quicker (even if not so precise) understanding.

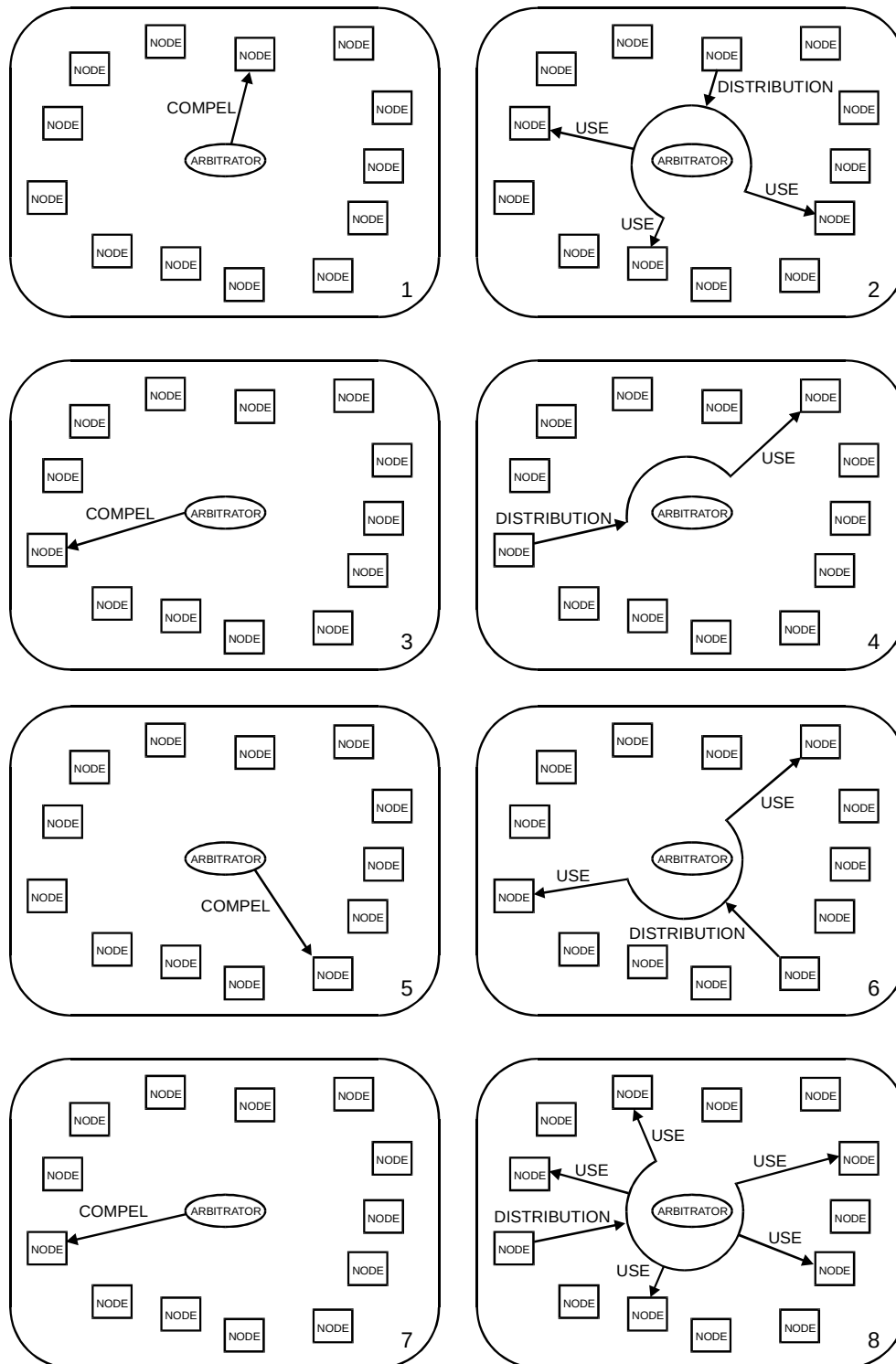
So, having to keep an "arbitrator" for the scheduled traffic, but wishing to freely circulate a token too, the only way is to (see figure A3)

- circulate the token only when no scheduled traffic is needed;
- make the token return to the arbitrator instead of passing it onto a new node so that the arbitrator, depending on the time left, can decide whether to actually pass the token once more or to resume link control to manage the scheduled traffic; and
- pass the token for a limited amount of time that is always shorter than the interval left before the next scheduled traffic.

Such an arbitral node takes the name of **Link Active Scheduler** (LAS, see A1.2.1) within the ISA Fieldbus DLL documents. That is,

"Link" because

the ISA fieldbus network can be made of several segments (links) interconnected by bridges (see clause A9 for the reasoning behind much of the multilinks structure). Each link has its own Active Scheduler that manages the relevant traffic independently from that of the other links; a node (bridge), which interconnects different links, stores the messages, addressed to links that can be reached along the network through it, and forwards them only when an opportunity is given on the addressed (or intermediate) links. This does not preclude having scheduled traffic over a multilink structure, but, in this case, a schedule builder must coordinate the single schedules of the different LAS of all the links. Such schedule builder could also run off-line and then download, through Fieldbus Management Services, the several pre-established schedules to each LAS.

**Figure A1 — Scheduled access**

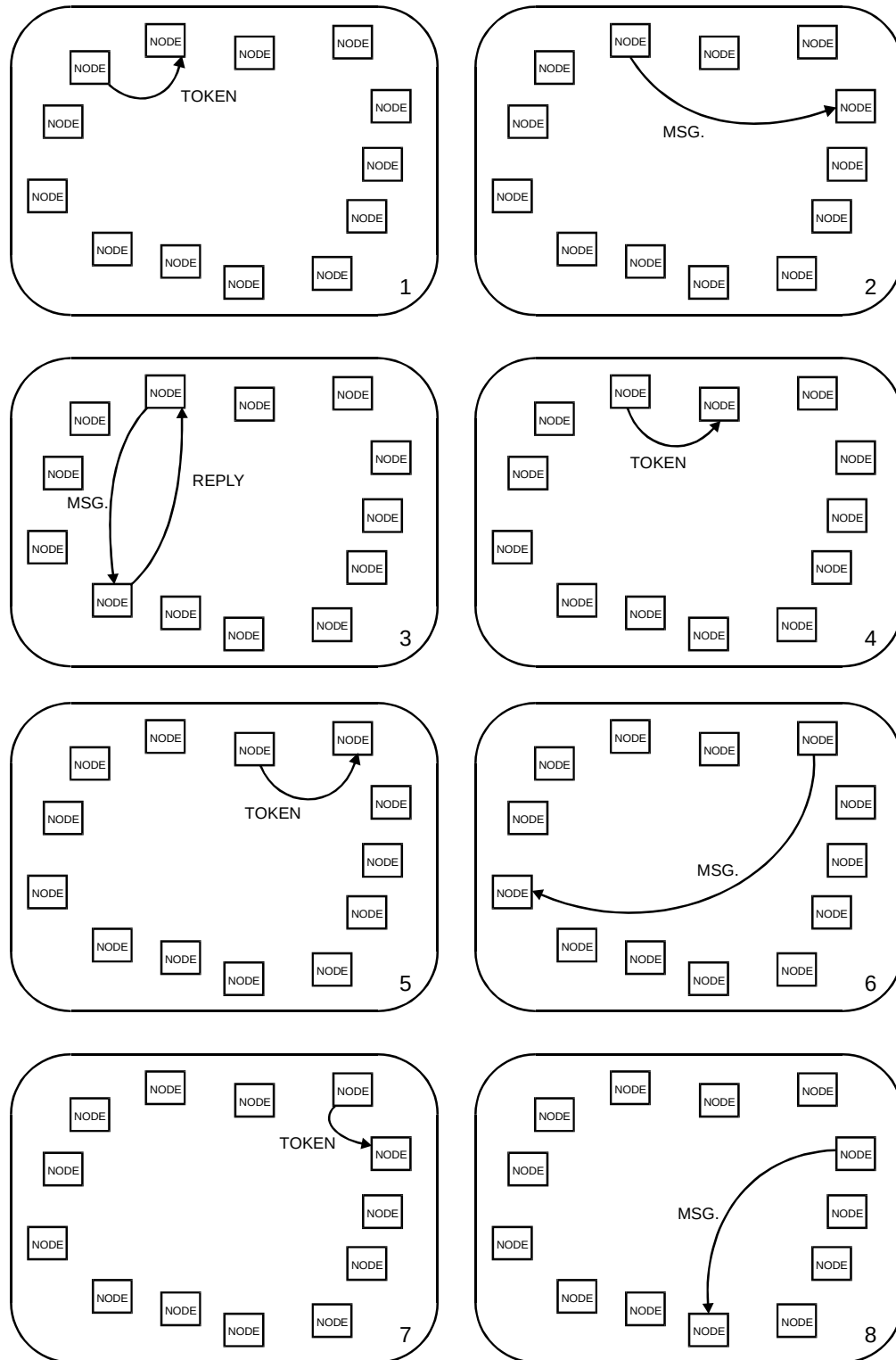


Figure A2 — Circulated token

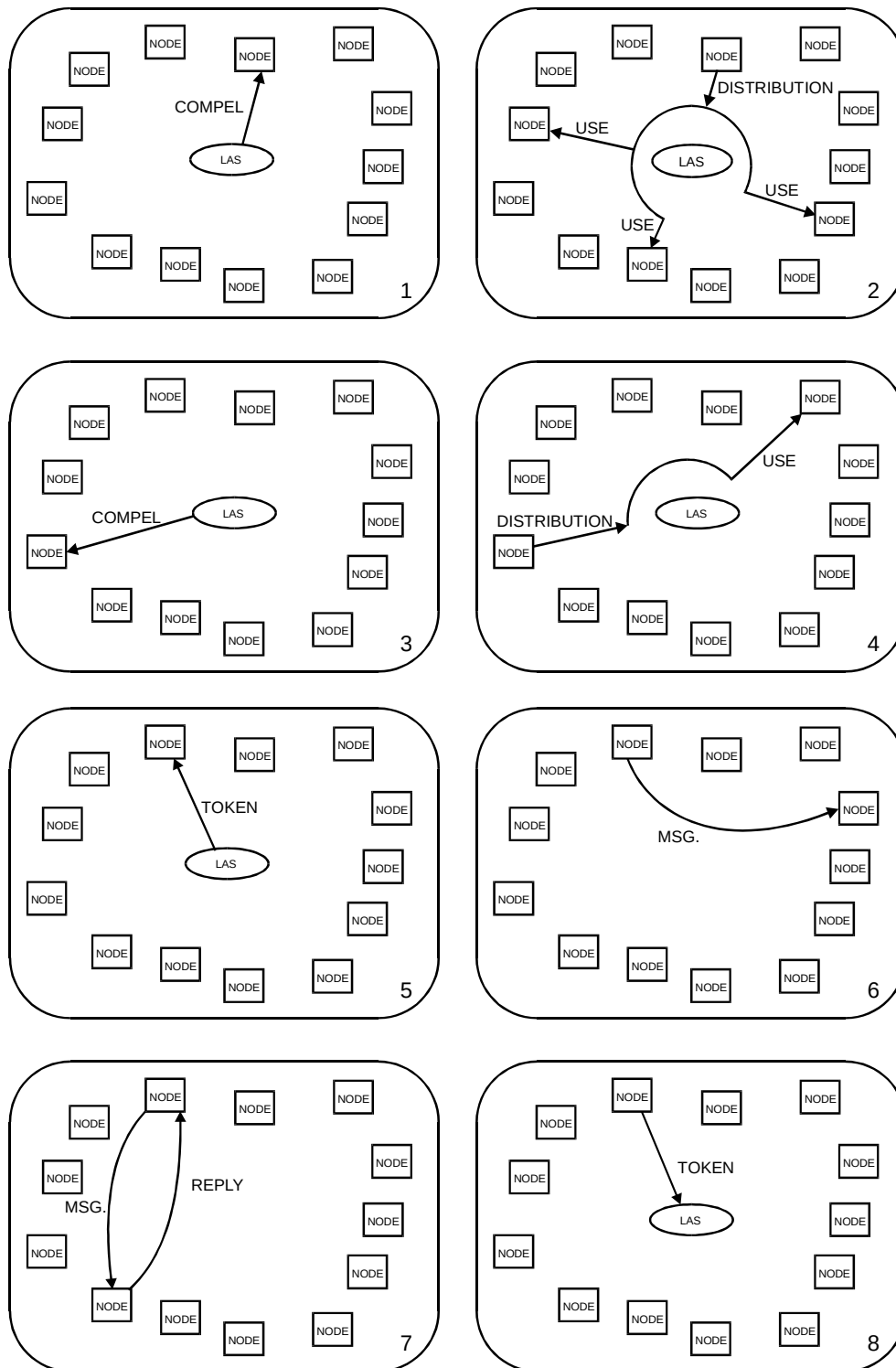


Figure A3a—ISA solution

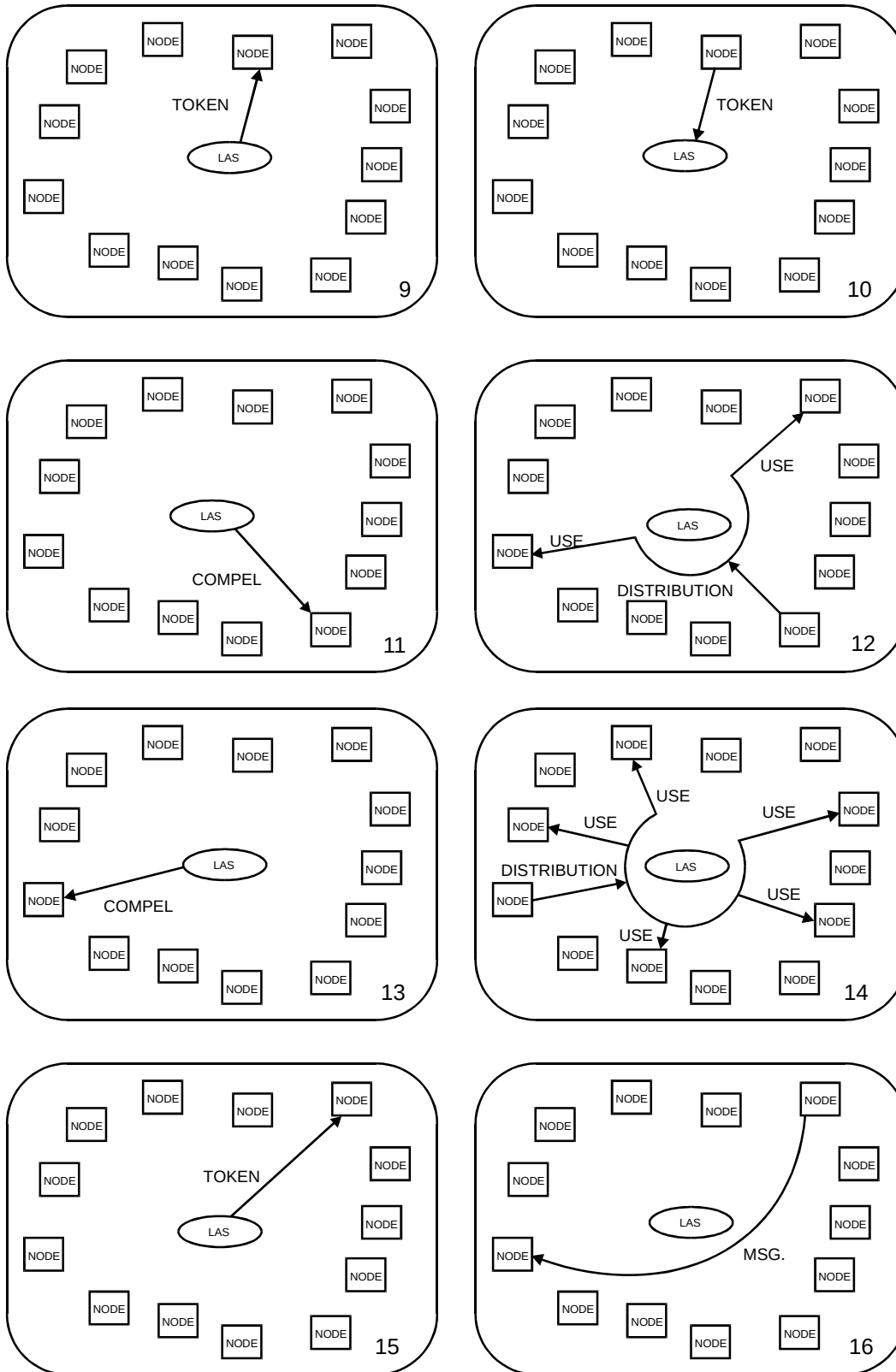


Figure A3b — ISA solution

"Active" because

on each link there can be more than one potential LAS even if, of course, only one at the time is the actual arbitrator; in fact, any one of the nodes belonging to the so called Link Master (LM) class can be elected as LAS (see A1.2.1.1) at the power up of the link, or when the current LAS fails, or when some new nodes are added to the link itself.

That is the main difference between LM class and the other "Basic" class, the possibility to become LAS. In the ISA Fieldbus DLL documents, belonging to the Basic Class does not mean not being able to receive and hold the token, nor does it mean necessarily being a "slave" of some "master" to which we have to reply; a Basic Class node just cannot become LAS, but it can instead make full and free use of the token for the duration the token itself is given to it (PIV-5.6).

Of course, the possibility to have more than one potential LAS on each link is essential for fieldbus availability reasons, see A1.2.1.1 and A13 for details on the LAS back-up procedures.

It is also possible that the several potential LAS have different levels of capability in managing and/or updating their schedule (see A1.2.1.4). This could be used, for instance, to provide the essential part of the fieldbus traffic in front of failures (soft degradation).

"Scheduler" because

the real activity of such an "arbitrator" is to strictly follow a defined schedule in order to decide (see A1.2.1.2) whether at that given time a scheduled action has to be triggered or whether there is a long enough gap, before the next scheduled action, so that a token can be given to another node (for a consequent maximum duration) or some link management activities (see A1.2.1.3) can be performed.

With relevance to all these scheduling and tokens managed by the LAS, specific items such as

- DLL services (DLL actions aimed to satisfy the DL-user), which can or cannot be compelled;
- scheduled and free data traffic;
- sequences; and
- token types and policy.

will be highlighted and described in the following subclauses.

First, as anticipated in the foreword, after going through these descriptions, readers looking for further refinements may find a graphic representation of the relationship among these items and between them and the DLL frames useful (Data Link Protocol Data Units [DLPDUs], see A5). Such graphic representation is found in B2, together with the relevant queues and buffers resources (see A2).

In any case we have to notice how,

- for the LAS to be able to precisely drive all the scheduled traffic,
- for the LAS and any other node to correctly handle the duration of the token use,
- to let any node ask the LAS for a defined action to be scheduled,

a precise enough sense of time and a relevant fine synchronization mechanism among all nodes are needed and hence provided by the ISA Fieldbus DLL documents.

In fact

- a precise DLL time is needed by the LAS to know "when to do what";
- a fine DLL clock frequency alignment is needed by the LAS and by each node that receives the token in order that they can both measure the duration of the token use at the same rate; and
- a DLL time synchronization among all the nodes is needed so that any node can request the LAS for a scheduled action to be executed at a defined "time" that represents the same absolute instant.

Clause A6 describes each mechanism of time distribution and synchronization. Here we would only like to highlight how all DLL recovery timers (a kind of time-out), to simplify implementations, are based on the same measurement unit called "slot-time" (PIV-3.3.25).

For each link, such slot-time is the measure (or safe estimation), maximized across all pairs of nodes on that link, of the worst asynchronism within a node between the perceptions of a defined basic DLL event and of its potential consequence generated by the other node (see B3 for details).

The slot-time is then used as common measurement unit (or granularity) of the DLL recovery timers over all the nodes of the link in the same way as several shopkeepers, having scales with different resolutions on the displays, decide to use the poorest resolution among all the scales as the common measurement unit in order to have numbers representing their weights with the highest resolution, but still meaningful for all the shopkeepers.

A1.2.1 LAS

A1.2.1.1 LAS election

Given the fundamental role of the LAS, it is of course essential to have a safe, univocal but fast enough procedure for electing the LAS itself. Would we like to have a war or endless voting when a new "king" has to be democratically and quickly appointed? *There are countries that are still suffering for such kind of hesitation, even when it's just a prime minister to be elected instead of a "king"!*

Here we are describing the procedure that, for instance, takes place when a link is powered up, or similarly when the current LAS fails, in order to decide which LM will then act as LAS. Other procedures apply when an LM asks the current LAS to relinquish its role and/or when the current LAS wishes to resign, but that is relevant to further refinements to optimize the schedule handling and updating (see A13); here we would like just to go through the main feature that lets any LM take over the LAS role so as not to make a dangerous bottleneck of the LAS itself.

Such a feature takes advantage of a three-step procedure that relies on the detection of link inactivity for a long enough period and on the use of a specific DLPDU called "Claim LAS" (CL) (PIV-7.19).

Step 1

In fact, **after detecting link inactivity for a period of $V(TN)$ slot-times (where $V(TN)$ is its node address), an LM sends a CL DLPDU.**

Now

- because node addresses cannot be smaller than 16; and
- because all the other DLL recovery timers are shorter than 16 slot-times (see A1.2.1.2, recovery timers after immediate reply requests [compel data, immediate acknowledge, two-way data exchange] or token delegation)

that means $V(TN)$ slot-times of link inactivity is a period long enough to infer that the current LAS had a problem or that no LAS existed so far.

Moreover, such a procedure makes any LM wait for a different period of link inactivity ($V[TN]$, being the node address, is different for each node), and this helps differentiate the claims of the several LMs when the current LAS fails. Then only the LM with the lowest node address will send a CL DLPDU; all the other ones, receiving such CL DLPDU, will stop their claiming procedure because of the link not resulting inactive any more.

Step 2

But, due to noise problems or to different switching up times of the LMs when the link is powered-up, we could still have two or more LMs sending their first CL DLPDUs at the same time.

That's why the procedure says that an LM, after sending that first CL DLPDU, has to wait for another $V(TN)$ slot-times period of link inactivity; if no other link activity is heard during this period, then that LM will send a second CL DLPDU.

In this way, even if two or more LMs sent their first CL DLPDU at the same time, only one of them will send another CL DLPDU because

- either the first CL was heard as link activity and so restarted the claiming procedures of those LMs from the same starting time; or
- the first CL was heard just as noise; but the second waiting period will differentiate those LMs.

Moreover, if an LM "x" is switched-on after the first CL DLPDU had been sent by another LM "y" having a higher node address, then three possibilities can happen

- either x hears a long enough link inactivity to send its first CL DLPDU before y sends its second one and so x will take the LAS role instead of y; or
- y will send its second CL DLPDU before x hears a long enough link inactivity to send its first CL and so y will carry on its claiming procedure stopping x from doing it; or
- x will send its first CL DLPDU while y sends its second one and so on.
- either such CLs are heard as link activity and so they restart the claiming procedures of x and y from the same starting time (then x will win); or
- such CLs are heard just as noise and then y will carry on its claiming procedure stopping x from doing it (see step 3 below).

Step 3

The procedure still requires the LM that sent the second CL DLPDU to verify that the link stays inactive for another period, randomly chosen in between 0 and 3 slot-times, before deciding to assume the LAS role.

This random period

- helps to avoid conflicts in the case of LMs having been configured, by mistake, with the same node address and not detecting each other's previous CL DLPDUs as link activity, but

- does not reopen the LAS election procedure to other just switched-on LMs (in fact, this period is shorter than any first waiting period to be followed, in its LAS claiming procedure, by a new node started just after the second CL of the successful LM).

A1.2.1.2 What an LAS can compel

In A1.2 we said that the LAS is a kind of king who tells everybody else what to do and when to do it. But it also gives everybody some defined amount of time for their free chats.

In the end, what exactly does the LAS compel and what does it leave free?

As anticipated in A1.2, an LM, after becoming LAS, assumes the right (called "**scheduler token**") to check if its current schedule requires any specific action to be taken at that precise DLL time (plus/minus a jitter optionally permitted by the schedule itself).

If so, the LAS triggers such an action, which could be either the delegation of the token to a node strictly for its execution of a defined sequence (see A1.2.2) or the compelling (**reply token**) of the transmission of one and only one message to be sent by some remote node; this latter message could then be

- either (most frequently) the conveyer of the value of a buffer associated with a DL-connection (see A2.2), or
- only in case of on-line updated schedules, the consequence of the completion, by the remote node, of an already issued DL-request, relevant to a DL-connection, whose execution had been deferred.

If, instead, there is enough time before the next scheduled action, the LAS uses this time to offer rights to access the link (**delegated token**) to some other nodes for a length of time and in accordance with a "rotation" policy that are both detailed in A1.2.3 and that, in any case, will never conflict with the respect of the next scheduled action to be issued by the LAS. Following this, if time is still available, the LAS will take care of sending some link maintenance traffic (see A1.2.1.3).

During the length of time any node holds a delegated token, it can perform any activity permitted by that token type and compliant with its LM or Basic Class (see PIV-table 1); in particular, for these tokens, such node can only

- provide other nodes with a reply token (that again means a token valid for only one message to be sent back). In this case, the range of possible transactions is wider (see A1.2.2 for details) than that one of the reply token directly given by the LAS.
- give the delegated token back to the LAS when no more useful delegated time is left or when no more activities are pending.

Any time a token is given to another node (either a delegated token by the LAS or a reply token by any node), a recovery timer is started by the token giver to check whether the receiver will start to make use of such token within defined time constraints. Otherwise the giver will resume activity (recover) before the other nodes assume an LAS failure (see in A1.2.1.1 the link inactivity period that triggers an LAS role claiming procedure).

Such recovery timers (PIV-3.3.26/27/28) will measure

- an immediate-response-recovery-delay (IRRD) immediately after giving either a reply or a delegated token; the recovery procedure at the end of this delay will avoid a deadlock (and avoid falling into a new LAS election) when the receiver of such tokens has any problem in receiving or using them; and

- a token-recovery-delay (TRD) during the normal link operations performed by a node that holds a delegated token; the recovery procedure at the end of this delay will let the LAS resume its scheduling activity (and avoid falling into a new LAS election) when the node that holds a delegated token has any problem in keeping alive the link traffic or in giving the token back to the LAS.

For each link, the values of these two delays are given starting from the definition of the "maximum-response-delay" (MRD), which is the worst case of delay (maximized over the link) introduced by a node before it initiates transmission of a DLPDU after having received either a delegated or a reply token.

Now, the given limits are

- **MRD** shall have a value of 1 to 11 slot-times;
- **IRRD** shall be $\text{MRD}+1$ slot-times; and
- **TRD** may have any value between $\text{MRD}+3$ and 14 slot-times.

It is important to note how these delays, MRD, IRRD, TRD and the waiting period for the LAS claiming procedure, are all in progression to enable the orderly succession of the different recovery procedures in accordance with the importance of their impact.

In addition, a special procedure is also provided (see PIV-6.2.7.1), for a potential extra slot-time monitoring at the end of IRRD, in order to avoid a noise near to the end of the IRRD period being seen as the beginning of the expected immediate response and causing the failure of the recovery procedure.

A1.2.1.3 Different types of traffic induced by the LAS

Beside the messages transmission (see A1.2.1.2) which the LAS triggers from, or permits to, other nodes, there is a set of messages that the LAS directly transmits (B14).

Let's see them by following the priority order of the main LAS overall selections (PIV-10.2.1)

- a) trigger any scheduled transaction first
- b) **if** no transactions are scheduled at this DL-time, then, without exceeding the gap before the next scheduled transaction, try to transmit
 - 1) Time Distribution (TD) DLPDU) if so required (see A6)
 - 2) **if** the current cycle of circulating the token among the link's nodes is not complete yet, then
 - a) proceed in circularly delegating the token by means of PT DLPDU (see A1.2.3) (and/or provide the extra Execute Sequence DLPDUs required by Request Interval DLPDUs, see a2 type sequences in A1.2.2)
 - b) **if** the remaining gap is not enough for a PT DLPDU transaction, then, without exceeding the gap before the next scheduled transaction, try to
 - 1) transmit a node-activation SPDU (see A8), if pending
 - 2) transmit a probe node PN DLPDU
 - 3) transmit DLPDUs waiting for transmission from address 0400hex (LAS address)

- 4) transmit Idle DLPDUs to fill the remaining gap in order not to leave the link inactive for over TRD
- 3) **else**, when the LAS has completed the current cycle of circulating the token among the link's nodes, then
 - a) **if** the LAS needs to transfer its role, then transmit all DLPDUs waiting for transmission from address 0400hex (LAS address) and then send a Transfer LAS (TL) DLPDU
 - b) **else**, without exceeding the limit V(LTHT) (see PIV-5.7.5.9) given to the link maintenance activities in one cycle of circulating the token, try to
 - 1) transmit a node-activation SPDU, if pending
 - 2) transmit one Probe Node (PN) DLPDU if none has been already sent within this cycle
 - 3) transmit all DLPDUs waiting for transmission from address 0400hex (LAS address)

A1.2.1.4 LAS scheduling profiles

The ISA Fieldbus DLL documents provide a very rich range of possibilities regarding the types of schedules an LAS can handle (see A1.2.2) and their possibilities to be on-line modified or to be updated from/to other LMs.

This has often made several people think that the ISA Fieldbus DLL proposal is expecting too much from such poor LAS; they know how many applications are needed to justify complex and expensive LAS, so they wonder how to satisfy their specific requirements with just a simple LAS.

In reality, for each LM node, the ISA Fieldbus DLL documents already provide a whole set of different profiles-as-LAS within which it is easy to find that one which better fits our application needs (PIV-11.3.c).

In decreasing order of complexity, these profiles are

"DYNAMIC", when the LM can support time-based pre-constructed schedules, can include modifications due to other nodes' sequence scheduling requests (when realistically feasible), and can perform current schedule transfer from/to other LMs.

"SHARABLE", when the LM can support time-based pre-constructed schedules and current schedule transfer from/to other LMs.

"UPDATABLE", when the LM can support time-based pre-constructed schedules and current schedule reception from other LMs.

"STATIC", when the LM can support time-based, pre-constructed by DL-management, schedules.

"NONE", when the LM can support only free-running (that means no time-based), pre-constructed by DL-management, schedules.

In other words

"DYNAMIC" means the LM is able to handle schedules whose execution is triggered by DLL time, is able to build new schedules on-line, is able to receive the current schedule from the LAS and to transfer its own to the other LMs.

Such an LAS can respond to any DL-user needs for on-line schedule construction and revision.

"SHARABLE" means the LM is able to handle schedules whose execution is triggered by DLL time, is able to receive the current schedule from the LAS and to transfer its own to the other LMs, but is not able to build new schedules on-line.

Such an LAS can satisfy any link scheduling, but the link needs a "DYNAMIC" LAS to update its schedule if we want to modify it on-line.

"UPDATABLE" means the LM is able to handle schedules whose execution is triggered by DLL time and is able to receive the current schedule from the LAS, but is not able to build new schedules on-line nor to transfer its own schedule to other LMs.

Also such an LAS can satisfy any link scheduling features and it needs a "DYNAMIC" LAS to update its schedule if we want to modify it on-line; but it cannot be used when we want to pass any on-line schedule change to other LMs.

"STATIC" means the LM is able to handle schedules whose execution is triggered by DLL time but only when they are pre-constructed by DL-management.

Such an LAS can satisfy any link scheduling, but only if this scheduling can be pre-defined and does not need to be changed on-line or transferred.

"NONE" means the LAS is able to handle only schedules that are pre-constructed by DL-management and executed continuously (free-running) without any reference to DLL-time.

Such an LAS would be better named "Link Active Sequencer" meaning that it triggers activities in sequence, one after the other, regardless of the current DL-time.

A1.2.2 Sequences

In A1.2.1.2, we have already mentioned how an LAS scheduled action may be either a direct compel for one specific action to be taken by a specific node or a token for a specific **sequence** to be executed by a specific node.

We have also said in A1.2.1.4 that some LAS profiles can modify schedule due to other nodes' scheduling requests relevant to some **sequences**.

What then are such sequences?

And how can the DL-user define their scheduling?

Let's start from this latter mechanism that is a significant DLL capability and, we believe, represents, from the OSI perspective, a non-trivial contribution by the ISA Fieldbus DLL documents.

In fact, due to the OSI principle of layering, once the DL-user has issued a primitive request, it could no more control the way of execution of such a request by the DLL, being that such an execution is internal to the DL layer itself.

How then can the DL-user control the scheduling that is just the timing of the primitives' executions?

Well, the ISA Fieldbus DLL documents introduce a Quality of Service (QoS) parameter "**DL-scheduling-policy**" (PIII-8.4), which, for any service (see B4) dedicated to transfer some user data, can defer a request issued by the DL-user until a specific DL-COMPEL-SERVICE request tells the DLL to take care of it.

It is like booking an open data journey at a travel agent and deciding about the departure day only at the moment of leaving. We do not interfere with the travel agent procedures for actually providing our means of transportation; we just agreed earlier about all the journey details (except its date) and "suspend" its ticket until the last moment. *Note — The travel agent has also confirmed, since the beginning, that the journey was possible.*

This DL-scheduling-policy parameter can take two values

"**IMPLICIT**", when a request is satisfied by the DLL as soon as possible, and

"**EXPLICIT**", when a request is deferred until a relevant DL-COMPEL-SERVICE request will be issued.

Now, this is the basic mechanism for scheduling but, once more, what about sequences?

Well, a DLL service called DL-SCHEDULE-SEQUENCE (PIII-29.3/4/5), lets the DL-user define and identify sequences just as lists of DL-COMPEL-SERVICES requests whose execution can be chosen to be either one-time or periodic or repetitive (for this last type see A1.2.4).

For one-time and periodic types, the DL-user can then define all the DL-time constraints (starting, cycle, etc.) and the permitted jitter of any so defined sequence. In fact, as a further major contribution to time integrity, **all** DLL actions have a fixed maximum execution time that is known in advance and no action may be started if its completion will exceed the available, scheduled time. So, unlike other protocols, message size variation and recoveries do not cause schedule jitter.

In this way, by means of using such constrained lists of DL-COMPEL-SERVICE requests, the DL-user can "direct" the timing (up to its feasibility) of all his requests of transferring data, a bit like an orchestra conductor who could first provide all the players with their instruments and then indicate when and how to play them.

That's also because the DL-COMPEL-SERVICE primitive has a very detailed set of actions that can be performed by using it (PIII-29.2) and that go quite beyond the simple release of a previously deferred local request. In fact, under certain conditions whose details must be read in PIII-29.2, it can also

- release a previously deferred remote request issued on a DL-connection;
- publish the value of a local or remote buffer associated with a DL-connection (see A2.2); and
- initiate a two-way exchange data transfer (see A3.4).

As anticipated, for people who are carefully reading the ISA Fieldbus DLL documents, some diagrams of B2 give more details about the way through which the DL-COMPEL-SERVICE (see PIV-9.4.2) acts on **local deferred DLL services**, on **remote buffers or deferred requests** and on **local buffers**.

The basic elements for the DLL to handle the **local deferred services** are the protocol "user-request queues" Qx(UR) (see PIV-5.7.3.1 and PIV-3.3.33), one for each user endpoint x, within which 3 areas can be distinguished

- area 2 is relevant to not-deferred requests;
- area 3 is relevant to deferred requests waiting to be "compelled" (at that moment they will be moved to area 2); and
- area 1 is relevant to requests already satisfied but still waiting for some acknowledgement.

Any time a request is put or moved into area 2, a reference to its Qx(UR) is appended to a protocol "unscheduled-service queue" of the node, Q(US) (see PIV-5.7.1.17 and PIV-3.3.34). Such Q(US) then acts as an ordered memo list for all the transactions to be performed by a node when a delegated token is (circularly) given to the node itself.

The basic element for the DLL to compel the publishing of **remote buffers and the release of deferred remote requests**, both associated with DL-connections, is the Compel Data (CD) DLPDU (see PIV-7.5) originated by a DL-COMPEL-SERVICE specifying the remote endpoint. When this CD DLPDU is received by the remote node it causes, as an immediate (on the local link) reply, the transmission of the buffer value or of the data originated by the released request (if this value and data imply more than one segment [see A11], then a reference to the relevant Qx[UR] is also appended to Q[US] in order to complete all the other segments transmission when the delegated token will be received).

The basic protocol mechanism for publishing the value of a **local buffer** associated with a DL-connection is simply the addition, within Q(US), of a reference to the Qx(UR) relevant to the buffer. Again, when the delegated token will be received and consequently Q(US) searched through, this reference will cause the DLL to form a DT DLPDU containing the buffer value and to publish it.

(The protocol mechanism for initiating a two-way exchange data transfer will be detailed within its specific section [A3.4], which also shows the reasons for such a feature.)

The same diagrams of B2 also detail the ways used by the ISA Fieldbus DLL documents in order to provide the execution of the above mentioned sequences.

Sequences can be mainly divided in two subsets from their execution point of view (see PIV-9.4.3)

a) those locally executed by the requesting node itself, e.g.,

- 1) sequences made of just one non-periodic compel to be performed as soon as possible; and
- 2) sequences different from a) 1) and generating DLPDUs some of which are not of CD type.

(We do not talk about repetitive sequences because they will have a clause on their own [A1.2.4].)

b) those whose execution is delegated by the requesting node to the LAS, e.g.:

- 1) sequences that, like a) 2) ones, are different from a) 1), but which generate DLPDUs that are all of CD type

Well, subset a) 1) is executed by appending to Q(US) a reference to the sequence's record itself so that, when the delegated token will be received and consequently Q(US) searched through, this reference will cause the DLL to form the DLPDU required by the single element of the sequence.

In the case of subset a) 2), an identifier is associated to the sequence record and an SPDU (Support Protocol Data Unit), requiring the LAS to provide space (within certain time constraints) for such a sequence, is prepared within QN(UR) (the Qx[UR] relevant to the node address) and transmitted through the now well known mechanism of appending to Q(US) a reference to QN(UR) waiting for the usual delegated token.

The LAS, when it receives such a request, will include it (if feasible because of its constraints) within the overall schedule so that, when the time arrives, an Execute Sequence (ES) DLPDU (see PIV-7.16) is sent to the requesting node specifying the identifier of the sequence to be executed and the amount of time assigned to such execution.

The node receiving the ES DLPDU executes the sequence and when that is finished it gives the token back to the LAS by means of a Return Token (RT) DLPDU (see PIV-7.17). If the amount of time delegated by the ES DLPDU was not enough to complete the sequence execution either because

- the sequence has been declared interruptible so that it can be partly done if the LAS did send only a portion of the total time needed to complete it (that's in order to easier fit the ES within the overall schedule); or
- because the sequence definition permitted dynamic appending of new compels; or
- because noise caused retransmission of some DLPDUs so extending the duration of the sequence.

Then the node gives the token back to the LAS by means of a Request Interval (RI) DLPDU (see PIV-7.18) through which it requests the LAS to find space within the overall sequence, for a specified additional time.

Note that, while an RT DLPDU may be sent to give the token back to the LAS, a faster procedure may also be used by setting a specific "F" bit (see PIV-6.2.1.2) within the Frame Control field of the last DLPDU sent during this period of token delegation. Such bit will tell the LAS that it has to resume the token.

In the case of sequence subset b) 1), an SPDU, requiring the LAS to send (within certain time constraints) all the CD DLPDUs requested by the sequence itself, is once more prepared within QN(UR) and transmitted through the usual mechanism of appending to Q(US) a reference to QN(UR) waiting for the delegated token.

The LAS, when it receives such a request, will insert it (if feasible because of its constraints) within the overall schedule and directly execute it when the time arrives.

Of course, all the described compelling, sequencing and ordering mechanisms, beyond the above mentioned protocol queues, also make use of data buffers and queues (already shown in the same diagrams of B2); they are detailed in the dedicated section A2.

A1.2.3 Token policy

During the development of the ISA Fieldbus DLL proposal, due to the continuous up and down of the preference between circulated token and scheduled access, some confusion often showed up regarding the way of satisfying traffic time constraints.

In fact, in classical solutions that **only** include a circulated token access mechanism, the mechanism itself is ruled by several deadlines and different priorities in order to provide a compromise among the following needs:

- keep the token cycle short enough (in order not to be too late in answering a sudden request)

and, at the same time,

- give enough contiguous time to each node, at least adequate token cycles (in order to also let it complete a worst case transaction)

and, at the same time,

- guarantee the most regular period of the token cycle among all the nodes (in order to also use the token cycle itself as a kind of rough application sequencer)

and, at the same time, always,

- satisfy the occurrence of high priority events (different token cycles for different priorities)

This set of token constraints is not appropriate in the ISA Fieldbus DLL proposal because the scheduled access mechanism already provides many of these classical needs

- specific and differentiated compel cycles made as short as required by the user in order not to be too late in answering a sudden need
- pre-check whether enough contiguous time for a specific transaction (sequence) can be provided
- a low as required jitter for any compel cycles
- all the most open choices of different compel cycles periods, one for each different urgency of events scanning resolutions

In other words, if a guaranteed time constraint is requested, then the scheduled access applies and takes care of it in the most satisfying way; if instead we don't want to permanently allocate bandwidth for some events that happen only occasionally and that do not require any guaranteed time constraint, then a circulated token is available as a flexible and appropriate solution.

Of course some deadlines (see the algorithm below) have to be observed by the circulated token proposed within the ISA Fieldbus DLL documents, but that does not mean we have to provide an emulation of the scheduled access while using the circulated token. The two mechanisms are separate; they satisfy different needs and must never be misused trying to respond with one of them to the requirements that are the basis of the other one.

So, when the LAS delegates the token to another node because no scheduled activity was planned at that DL-time (see A1.2.1.3), it has to simply apply the following algorithm:

- Try to send a Pass Token (PT) DLPDU (see PIV-7.15) circularly to all the nodes connected to the local link and that wish to receive the token itself (by belonging to the token-circulation-list V[TCL]). Each PT frame specifies the maximum duration of its use that must be
 - no greater than

the gap remaining until the next scheduled activity minus the overhead in delegating the token itself; and

the maximum duration of token use configured for that node V(MTHA) (see PIV-5.7.5.10); but
 - at least as great as

the default minimum duration of token use configured for the link V(DMDT) (see PIV-5.7.5.7).

The node receiving the PT DLPDU uses such delegated token to perform the transactions listed in its Q(US) (see A1.2.2) and that have priority not lower than that specified by the PT DLPDU itself (the higher the priority, the shorter the permitted DLPDU's length

priority URGENT → DL-user data ≤ 64 octets,

priority NORMAL → DL-user data ≤ 128 octets,

priority TIME-AVAILABLE → DL-user data ≤ 256 octets).

When that's finished, or when the delegated token duration is expired, it gives the token back to the LAS by means of a Return Token (RT) DLPDU (see PIV-7.17). If the amount of time delegated by the PT DLPDU was not enough to complete what was within the Q(US), then the node gives the token back to the LAS by means of a Request Interval (RI) DLPDU (see PIV-7.18) through which it requests the LAS to resend the token to the same address for a specified additional time.

Note again that, while an RT DLPDU may be sent to give the token back to the LAS, a faster procedure may also be used by setting a specific "F" bit (see PIV-6.2.1.2) within the Frame Control field of the last DLPDU sent during this period of token delegation. Such bit will tell the LAS that it has to resume the token.

If the delegated token is sent back by an RI DLPDU, then the LAS will try to send again the PT DLPDU to that node during the same cycle of circulating the token, specifying the new maximum duration of its use, which must be

- not greater than

the gap remaining until the next scheduled activity minus the overhead in delegating the token itself; and

the remaining duration of token use for that node $V(RTHA)$ (see PIV-5.7.5.13) that is $V(MTHA)$ minus the amount of time already used by the same node during the current cycle of circulating the token; but

- at least as great as:

the additional duration of token use requested by that node in the RI DLPDU.

When the LAS circulates the token in such a way, if the gap that remains until the next scheduled activity is not enough to guarantee the above due duration of token use for a certain node, then that node can be temporarily bypassed in the list of nodes to which the token has to be delegated. This may optimize the use of the bandwidth because the duration of token use for some other nodes could instead fit within that gap.

But, in any case, any bypassed node has to receive its delegated token before the completion of the current cycle of circulating the token.

A PT DLPDU with duration equal to zero can only be sent to probe the presence of nodes that are connected to the link (so included in the live list $V[LL]$, see A8) but not presently wishing to receive any delegated token (they are not included in $V[TCL]$, see again A8).

At the end of any cycle of circulating the token, the LAS compares the actual period of token rotation $V(ATRT)$ (see PIV-5.7.5.12) with the target token rotation time $V(TTRT)$ (see PIV-5.7.5.11), and if the token went "slower" than required, then the priorities allowed by the next PT DLPDUs will be increased, if possible, to permit only shorter DLPDUs; if instead the token went "faster" than required, then the priorities allowed by the next PT DLPDUs will be decreased, if possible, to also permit longer DLPDUs.

It should not be surprising that the actual period of token rotation could result in longer than the target rotation time; in fact, it can be wise to fix a target rotation time shorter than the simple sum of the maximum durations of token use of all the nodes. That is because

- statistically, not all nodes will use the token for their maximum allowed duration within the same cycle; and

- *when, in spite of that, it happens that a consequent "optimistically" evaluated target rotation time is exceeded, then the permitted frames' priority will be immediately increased so opposing, for some time, a reduced maximum allowed length of frames to such a statistical exception.*

Note that to help the target token rotation time be achieved, the other types of link traffic can be limited. When the schedule of the scheduled traffic is constructed, it must not exceed a defined portion (see PIV-5.7.5.5) of the target token rotation time itself and the link maintenance traffic (see A1.2.1.3) cannot exceed a defined number of octets (see PIV-5.7.5.9) per cycle of circulating the token.

Finally, each time the LAS delegates the token, recovery timers (see A1.2.1.2) are started in order to avoid deadlocks due to any problems that could affect the node supposed to receive and use such token; that is always to keep the LAS master of the traffic over the link in order to avoid any perturbation to the scheduled traffic beyond the affected node itself.

A1.2.4 Repetitive sequences

As we mentioned in A1.2.2, sequence execution can also be chosen to be repetitive. That can be appropriate when in a node we have a set of transactions to be activated cyclically but we do not care about the period of such a cycle. The two requirements of such transactions are

- a) to be activated as frequently as possible; but
- b) without any guarantee about the regularity of the frequency itself.

A1.2.4.1 Why repetitive sequences are provided

The two above requirements could appear to be in contradiction. It may look like a boss telling the director of one of his subsidiaries, "You must always keep me updated about your business! But I don't care whether you do it every hour or every few years."

Strange, isn't it?

In reality, things are a bit different. Repetitive sequences are better represented by a boss telling that director of one of his subsidiaries,

"You must keep me updated about your business every time the company shuttle calls at your offices: it will never stay without calling for longer than a week. Always give a copy of your current results, even if unchanged, to the shuttle driver; he never has too much mail to carry."

It's from this sentence that we can get the true peculiarities and reasons of repetitive sequences:

"... the company shuttle will never stay without calling for longer than a week ..." Repetitive sequences require that the media access mechanism, even if without guaranteeing a constant period, provides a minimum lower bound of the updating frequency—the higher, the better but at least never less than a given value. We'll not be able to synchronize our watch on that frequency, but we'll never have to wait for too long.

"... the shuttle driver never has too much mail to carry." Repetitive sequences, just because executed as frequently as possible, can apply only when the amount of traffic (for that node or for the link itself if many nodes use repetitive sequences) is not so heavy as to saturate the bandwidth. Otherwise a mechanism attentive to the real needs of transmitting only new information would be more appropriate.

"... Always give a copy of your current results, even if unchanged, ..." Again, this can simplify the job of whom is sending information; he doesn't need to care about changes before deciding whether to send

the information or not. But, once more, it is suitable only if we are far away from saturating the bandwidth.

A1.2.4.2 Repetitive sequences within ISA Fieldbus DLL documents

ISA Fieldbus DLL documents satisfy the above criteria about repetitive sequences (see PIV-29.3) by executing them as frequently as possible, but no more than once per cycle of circulating the token among all nodes on the link. This feature

- a) if the node does not need more than its token time, provides a minimum lower bound of the repetitive sequence updating frequency (as noted in A1.2.3 about the period of token rotation); and
- b) leaves time to the traffic of the other nodes (and to other traffic of the same node).

This is achieved through the following mechanism:

- a) A repetitive sequence, defined by a DL-SCHEDULE-SEQUENCE request, is taken into account (see PIV-9.4.3) by appending to the protocol "unscheduled-service queue" Q(US) (see PIV-5.7.1.17 and PIV-3.3.34) a reference to the sequence itself.
- b) When the node receives the delegated token circulated by the LAS and, consequently, Q(US) is searched through (see A1.2.3), that reference will cause the sequence to be executed (see PIV-7.15.4.1.1b). After such execution, the reference is not removed from Q(US) but deactivated until the next cycle of circulating the token so meeting the criterion of "execution as frequently as possible, but no more than once per cycle."

A2 Buffers vs. queues

A frame sent by the DLL onto the medium, through the Physical Layer, is made of

- always, a part coded by the DLL itself (Frame Control: FC [see PIV-6.2.1], Frame Check Sequence: FCS [see PIV-6.1.1], DL-address[es] [see A4], DL-parameters specific of the DLPDU [see PIV-8]); and
- in most cases, a part passed by the DL-user and usually known as "DL-user data."

Such DL-user data has, of course, to be stored somewhere within the DLL while waiting for its transmission (or, on the receiver side, while waiting for the DL-user to read it).

To perform this storing, the DLL has to provide some kind of resources that, usually, can belong to two different big "families," queues and buffers. Let's see the main differences between these families.

A buffer-to-buffer data transfer reminds us of a camera-to-monitor transmission. At each new scanning, the view **now** in front of the camera updates the image shown on the monitor, regardless of whether that view is different or not from the previous one and whether someone had already looked to the previous image on the monitor. Similarly, in a buffer-to-buffer data transfer, at each new communication instance, the data value **now** contained within the sending buffer updates the previous one onto the receiving buffer, regardless of whether that data value is different or not from the previous one contained within the sending buffer and whether someone had looked to the previous one within the receiving buffer.

A queue-to-queue data transfer reminds us of a fax-to-fax transmission. The transfer can be activated only when some new pages are inserted within the sending fax. Those pages will then be placed by the receiving fax (at the back of some previously received ones, if still there) all at disposal of the receiver for when he decides to read them even if he will do that only once every so many transfers. Similarly, in a queue-to-queue data transfer, only when some new data are inserted within the sending queue the

communication can be activated. Those data will then be placed within the receiving queue (at the back of some previously received ones, if still there), all at the disposal of someone who wishes to read them even if he will do it only once every so many data transfers.

As a consequence of what was just said

a receiving buffer makes sense if

(the sender is always the same **AND** the data is always the same [only its value may vary])

(otherwise, how could we be sure the receiver does not lose something important from someone?)

OR

(the receiving buffer also guards each sender address **AND** the flow rate is slow enough to let the receiver read any new incoming message)

(otherwise, how could the receiver know where the data come from or be sure not to lose something important?)

a transmitting buffer makes sense if

(the data is always the same [only its value may vary] **AND** the receiver is always the same)

(otherwise, how could we be sure the sender does not miss sending something important to someone?)

OR

(the transmitting buffer also guards each destination address **AND** the flow rate is fast enough to let the sender transmit any new data)

(otherwise, how could the sender know where the data has to be sent or be sure not to miss sending something important?)

On the other end, a queue shall be used when different data have to be sent to different receivers, or different data have to be received from different senders, all regardless of the flow rate. In fact

a receiving queue is needed if

the sender is not always the same **OR**
the data is not always the same **OR** the

flow rate is not slow enough to let the receiver read any new incoming message

(so we can be sure the receiver does not lose anything important from anyone)

a transmitting queue is needed if

the data is not always the same **OR** the receiver is not always the same **OR** the flow rate is not fast enough to let the sender transmit any new data

(so we can be sure the sender does not miss sending anything important to anyone)

A2.1 Why both

So, when should we choose a buffer and when a queue?

When is a buffer better than a queue or vice versa?

And also, can we satisfy any communication need by using only queues or only buffers?

At the end of the previous clause we have seen when buffers make sense and when queues shall be used instead, but from that analysis it appears that queues could always be used to avoid introducing buffers at all. In fact, if there are cases which require queues and cases where buffers make sense but where queues do not make anything wrong, why should we use also buffers?

Well, from a practical perspective, in several applications (especially the very simple ones) all the communication needs could belong to the set where buffers make sense and this could lead to very simple implementations that use only (or mainly) buffers and do not include (or include at a very reduced level) data queuing mechanisms and the relevant memory allocation.

But also from a "communication" perspective, a buffer can be very useful when the system does not need any newly produced data value to be transmitted. For instance, if the application(s) of the receiving device(s) can cope with an updated value of a defined data every 100 msec, even if the application of the transmitting device produces a new value every 10 msec, it is useless (and bandwidth consuming) to actually transmit a new value every 10 msec. By using a transmitting buffer, we can leave the application of the transmitting device updating the value as often as it likes without requesting a correspondent transmission that takes place only when the schedule of the link, or the application(s) of the receivers, decides.

Similarly for receiving buffers, when the relevant updating, caused by the communication, is faster than some of the receiving applications and we do not want to trouble them in getting values not needed out of a queue, a buffer is the right solution that follows the last communication updating and keeps it available for when the receiving application needs it.

For all of the above, the ISA Fieldbus DLL documents provide both queues and buffers, in the several combinations and in accordance with the characteristics shown in the following clause.

A2.2 ISA buffer and queue features

The initial step provided by the ISA Fieldbus DLL documents, while dealing with queues and buffers, is the capability itself of creating such kinds of resources. Two specific services, DL-CREATE and DL-DELETE (see PIII-12.1 & 12.2) give the DL-user the possibility (also provided by DL-management) to

- allocate or de-allocate the necessary resources either for a queue of a defined depth "K" or a buffer that can be either non-retentive (empty each time after being read) or retentive (unchanged after being read);
- define the identifiers (the local references) of such a queue or buffer for both the DL-user and the DLL itself; and
- limit the maximum length of the single user data frame (Data Link Service Data Unit [DLSDU]), which can be contained by the buffer or of each one of the K DLSDUs, which can be contained by the queue.

These queues and buffers are often referred to as "explicit", because of the explicit user request to have them done; we'll see (in A3) how data transfer can also occur without explicitly creating such resources just by letting the DLL use its so called "default implicit OSI queues," a bit like when, people in a hospital cannot each pay for a private room (with dedicated features) and have to share a common ward (potentially unavailable because overcrowded by other people) (see also B2).

Buffers and queues, to be operative, have to be bound (associated) either to normal access addresses (Data Link Service Access Points [DLSAPs]) for connectionless services, or to "agreed" end-points (Data Link Connection End-Points [DLCEPs]) for connection-oriented services. We'll see how all this works in A3, now we wish only to underline how such resources can work after being associated with a defined end-point; for example, a mailbox will be correctly filled by the postman only after you put it by your street number plate or write your name on.

Queues and buffers have to be bound both on the transmitting side (sources) and on the receiving side (sinks) of a data transfer. On both sides they can have different numbers of inputs (from the DL-user, if sources, or from the network, if sinks) and outputs (to the network, if sources, or to the DL-user, if sinks). In general

- a queue can have more inputs but just one output (*otherwise, several "writers" can easily pile their different data, one within each entry of the queue; but after a "reader" gets out a data this is cleared and not available any longer for other readers*). (See figure A4.1.)
- a retentive buffer (R buffer) can have more outputs but just one input (*in fact, several "writers" would end in overwriting their different data within the same location; but it can easily happen that several "readers" need to get a copy of the same data*). (See figure A4.2.)
- a non-retentive buffer (NR buffer) can have only one input and only one output (*otherwise, all the readers following the first lucky one will find the buffer empty*). (See figure A4.3.)

Furthermore, over a data transfer

- a queue can only be emptied into another queue (*in fact, if we should try to empty it into a buffer, how could we be sure that the receiver DL-user reads a just arrived data before this one is overwritten by the incoming following entry of the queue?*). (See figure A4.4 and figure A4.5.)

- a buffer, of course, can be copied into another buffer (*because of the matching use of the two*) but also into one entry of a queue (in this latter case, going back to the analogy made at the beginning of A2, it is like connecting a camera to a fax via a scanner; it will keep memory of each single image). (See figure A4.4.)

A3 Connections vs. connectionless

This has certainly been another subject of hard battle among the experts of the ISA Fieldbus committee, and that is quite natural when two solutions face each other without one having solely advantages and the other one only disadvantages.

The same applies for connection-oriented versus connectionless communications. The first ones imply protocol mechanisms relying on many variables, states and resources plus some kind of initial (either on-line or pre-established) agreement (hence negotiation) between the parties, but can provide a very reliable, efficient and "rich of features" data transfer; the second ones can be implemented with simple protocol mechanisms but are quite poor in terms of "added value" to the basic data transfer and of keeping trace of the data transfer "history," which is essential to the DLL to be self-reactive to any possible consequence of some fault or noise over the data transfer itself.

Let's see a possible analogy.

It's like a well-built marriage compared to a suddenly-started free relation. Certainly the marriage is more complex and requires some understood agreement from the beginning, but it gives much more possibilities to overcome the bad moments due to the increased mutual knowledge usually pre-induced by the need of weighing the decision to get married.

(Don't worry, the ISA Fieldbus DLL documents, for their connections, also provide an easy and fast divorce, the DL-disconnection. And, for less serious situations, there's also a kind of Justice of the Peace advice, the DL-reset).

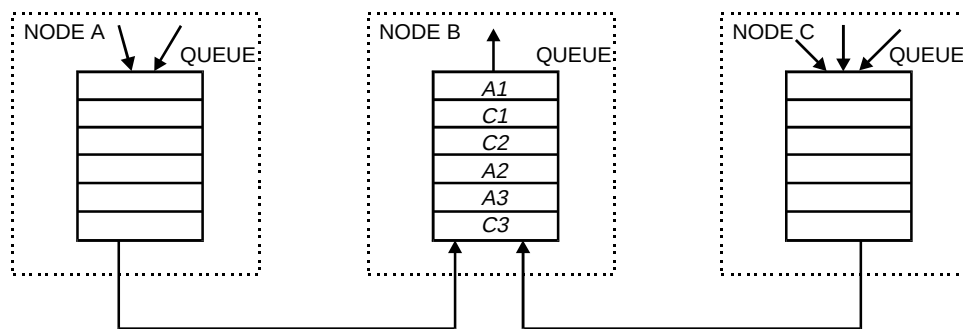


Figure A4.1

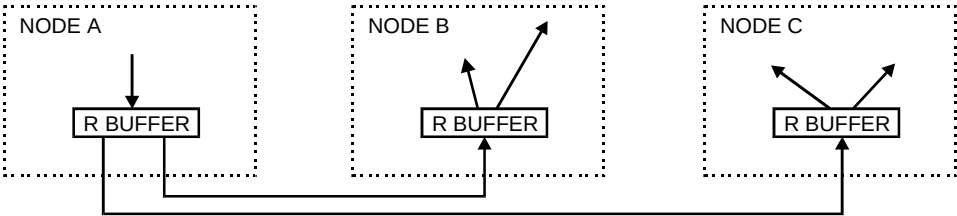


Figure A4.2

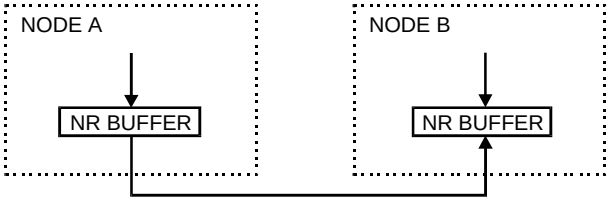


Figure A4.3

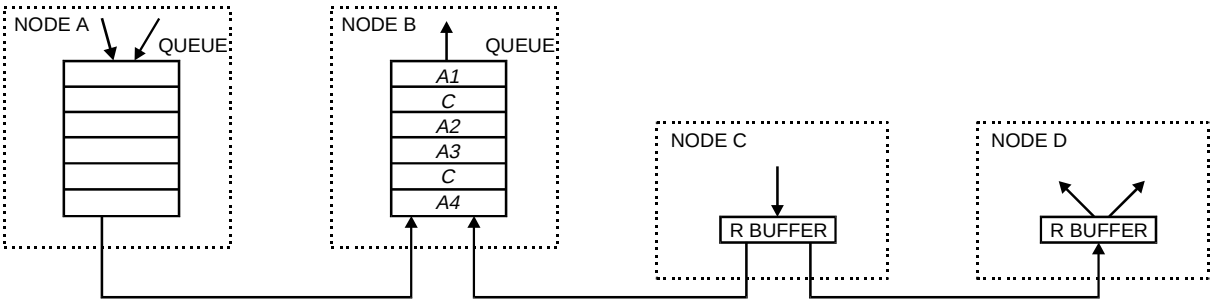


Figure A4.4