CANopen-based Distributed Intelligent Automation

K. Etschberger, C. Schlegel, IXXAT Automation

First the main benefits of distributed processing will be discussed and the requirements summarized, which have to be met for the implementation of intelligent distributed systems. With the additional CANopen standard DSP 302 a framework for distributed systems is now specified which, besides other items, comprises the introduction of network variables, the support of program download and control, a standardized system boot-up procedure as well as the definition of a configuration management instance. As system configuration is one of the most important practical aspects for the establishment of distributed intelligent systems, an introduction into the configuration process of those systems will be given and a versatile configuration tool shortly presented. Finally, it is shown how the implementation of programmable CANopen devices can be facilitated considerably on the basis of an available CANopen Master software package.

1. Benefits and Requirements of Distributed Intelligence

Distributed processing in automation systems today is still not widely in use. One reason is that popular fieldbus communication systems do not efficiently support multi-master communication. On the other hand, there are many advantages of distributed intelligence. CAN-based communication systems supporting producer-consumer oriented communication are well suited for the establishment of systems with distributed intelligence. The main benefits of distributed processing are:

• **Lower processing requirements and higher system performance, respectively:** Since in a system with distributed intelligence the overall processing task is distributed to a number of processing units, the processing load of each unit is considerably lower than in a system with only one central processing unit – or at the same processing capacity per processing unit, a much higher system performance is possible.

• **Higher modularity and flexibility**: In a well designed distributed intelligent automation system processing capacity it allocated to independent functional units or subsystems. Therefore adding and extending of further functional units or subsystems should be much simpler than in systems with central processing.

• **Easier system development, installation and testing**: Due to the implementation of the system in form of independent functional units or subsystems development, installation and testing of the independent functional units can be accomplished in parallel. Since the processing is limited to smaller subsystems, program development should be simplified as well and integration of the subsystems should be much easier than with a centrally controlled system.

• **Better real-time behavior**: Due to the local processing capacity even very time critical tasks, like closed-loop control can be performed on a system with distributed processing.

• **Lower bus load and lower bit rate requirement, respectively**: A system with distributed processing does not need to transmit each single signal or event via the communication system to a central control unit, since most of the processing is performed directly at the remote subsystems. This should result in a considerable reduction of the bus load or alternatively in a reduction of the bit rate.

• **Higher system availability**: If it is possible to organize the system into highly independent subsystems or functional units, the system can remain at least partially operational even when one subsystem or functional unit fails. In a centrally

controlled system, a failure of the central controlling unit causes the failure of the complete system.

Of course, distribution of intelligence is not without costs. Besides the non-availability of the appropriate technology certainly the higher costs of utilizing multiple processing units was one of the main reasons for the relatively minor usage of distributed intelligent systems until now. In addition, there are several specific requirements for distributed processing, such as:

- Globally (network-wide) accessible program variables
- Support of system configuration and programming by efficient tools
- Powerful network management functionality featuring plug-n-play capabilities
- Support of program download and remote program control

In the following, it is shown that with the extension DSP 302 of the CANopen standard a comprehensive framework for the implementation of distributed intelligent systems is provided; also efficient system configuration tools and universal, cost-efficient processing devices and I/O-modules based on DSP 302 are now available in the marketplace.

As an example, Figure 1 shows a coal conveyor plant, which is completely controlled by a CANopen-based automation system according to DSP 302. The system, developed and installed by the Helmut Mauell company [1], comprises 11 IEC-61131-3 programmable control units and 44 universal I/O modules. The complete system consists of more than 1000 data points and 280 network variables. More than 500 PDOs have been established for the transmission of process data and network variables more than 500 PDOs have been established.

Figure 1: Coal conveyor plant of the harbor power plant, City of Bremen/Germany, controlled by a CANopen-based automation system with distributed intelligence (Foto: Helmut Mauell GmbH).

2. CANopen DSP 302: A Framework for Distributed Intelligent Systems

The CANopen Communication Profile [2, 3] specifies the basic mechanism for exchanging process and service data via the CAN bus. The Communication Profile also describes the heart of the CANopen standard, the CANopen Object Dictionary (OD) which provides a standardized description of any communication and application parameters, data and functions of a CANopen device which are accessible from the CAN bus via so-called Service Data Objects (SDOs). Also services for network management, emergency messaging, the provision of a global system time and synchronization are specified in this part of the standard.

Within an additional framework for programmable CANopen devices, specified in [4], further functions are specified for the implementation of distributed intelligent systems. Main parts of this specification are the

- Definition of “Network variables”
- A mechanism for program download and program control
- A standardized system “Bootup-Procedure” as well as functions for configura-

tion management and the dynamic establishment of SDO-channels.

For IEC 61131-3 compatible programmable devices a detailed application profile is being specified in [5].

2.1 Network Variables

As the most essential requirement for distributed intelligence the interaction between application programs running on different nodes of a network must be possible. This means that an application program on node X can communicate directly with an application program on node Y just by reading or writing of a common variable. Of course, this variable has to be declared on both application programs as “remote” or “external” or, according to CANopen, as “Network Variable”. Therefore CANopen DSP 302 specifies variables which may be accessed via the network as a new type of application objects, to be described in the object dictionary of a programmable device. Since the usage of network variables cannot be specified before programming of an application, a programmable device only can provide the means for specifying of network variables in its object dictionary. Thereby the specific attributes of a network variable are its direction (input or output) and its data type. Network variables are mapped into the object dictionary in form of data direction - and data type -specific index-segments. DS 302 does not specify where in the index range of the object dictionary network variables are to be described. As an important type of standardized programmable devices DSP 405 [5] specifies this for IEC 61131-3 compatible programmable devices. There, for any of the specified data types 64 main-indices with each 254 sub-indices are available. This is shown in Table 1. A further introduction into this topic is given in [6].

Table 1: Mapping of input network variables into the CANopen Object Directory according to IEC 61131-3 compatible devices

Start Index	Data Type
A000H	Integer8
A040H	Unsigned8
A080H	Boolean
A0C0H	Integer16
A100H	Unsigned16
A140H	Integer24
A180H	Unsigned24
A1C0H	Integer32
A200H	Unsigned32
A240H	Float32
A280H	Unsigned40
A2C0H	Integer40
A300H	Unsigned48
A340H	Integer48
A380H	Unsigned56
A3C0H	Integer56
A400H	Integer64
A440H	Unsigned64

Figure 2 shows the relationship between a local application program with remote application programs via network variables. Remote or external input and output variables defined in a local application program refer to network variables located in the object dictionary of the local CANopen Interface. Network variables are denoted according to the direction as seen from the network. Therefore input network variables are seen as inputs to the network. This corresponds to outputs of the local application programs and inputs of the remote application programs, respectively and vice versa.

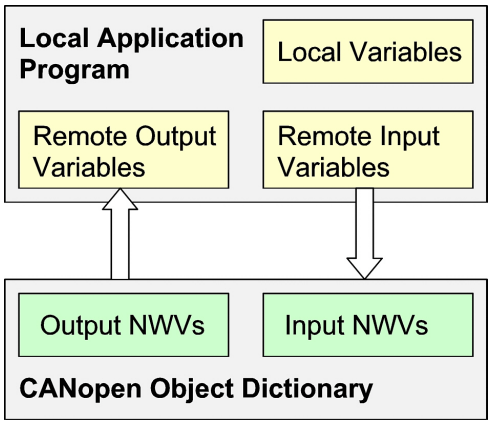

Figure 2: Definition of CANopen input and output network variables

Most programming systems support a mechanism for the definition of resources. This can be used to assign the CANopen attributes of a Network Variable like index/sub-index of the location in the OD and its associated data type to the corresponding symbolic name of a variable in the application program. Based on a user-friendly system configuration tool this can be accomplished by importing a so-called Device Configuration File (DCF) created as the result of the configuration process (See 3.1).

In order to allow programmable devices the usage of a so-called “process image” for input and output variables – a method which is especially common for PLCs - a relationship between the process image and the location of the network variables in the CANopen OD is necessary. The network variables storage capacity of a programmable device is described in the section “Dynamic Channels” of the Electronic Data Sheet (EDS) of the device. Besides the number of storage segments available, the data type, direction, index range, offset related to the base address of the process image and maximum number of objects which can be allocated are specified for each segment.

After configuration of a programmable device, i.e. allocation of the used network variables, the specified variables in combination with the information of the EDS is provided by the Device Configuration File (DCF) for usage in the application programming system.

2.2 Program Download and Control

Downloading of a complete application program or parts of a program, for example from a system configuration tool, may also be considered as a mandatory capability of a distributed intelligent system. To support this feature, a device needs to provide only a basic CANopen bootstrap-loader.
Downloading of the program code and data is supported by means of a standard SDO transfer protocol, e.g. the SDO block transfer protocol, by addressing the specific “Download Program Data” - object (Index 1F50H) in the OD. With up to 254 sub-indices the downloading of up to 254 different application programs or parts of them is possible.

For controlling of a remote application program via the bus, the “Program Control” – object (Index 1F51H) has to be addressed. By writing a specific control word to the corresponding sub-index starting, stopping or resetting of a specific program is possible.

With a third object (1F52H), the “Verify Application SW” – object, the latest date and time of an update of the application programs can be stored.

2.3 Network and Configuration Management

Network Management

The network management in a CANopen system is organized in form of a master-slave relationship between a node which acts as the master – this node is called a “NMT-Master” – and the other nodes which act as slaves (“NMT-Slaves”). Please note that the master/slave feature only refers to network management. Network management in the context of a CANopen system comprises the functions of controlling and monitoring the communication status of the nodes.

Since in such a system only one device can perform the NMT-mastership, but several devices may have the capability to do this, it is necessary to have the possibility to configure this functionality. For this purpose a specific “NMT-Start-Up” – object (1F80H) has been defined to specify the NMT-mastership as well as the specific start-up behavior of an assigned NMT-Master. It should be mentioned here, that there is also a pending standardization proposal for a “Flying-Master” feature for CANopen networks. It is intended that this feature will get an optional part of DS 302.

Standardized System Boot-up

As a further extension to DS 301, DSP 302 introduces a standardized system boot-up procedure to be performed by the NMT-Master during initialization of a CANopen system. To perform these functions, the NMT-Master needs to know what nodes (NMT-Slaves) are to be managed. This information together with further data is available to the NMT-Master in form of several network related objects.

The first of these objects, the so-called “Slave-Assignment”- object (1F81H) provides information about the slave nodes assigned to the NMT-Master, how they have to be booted (there are several booting options), specifies the different error control parameters and the requested actions in case of an error control event detected by the NMT-Master.

There are further data structures (objects) specified for checking the system identification such as Device Type, Vendor Identification, Product Code, Revision Number or Serial Number of any of the devices. This information is used by the NMT-Master to verify the correct system configuration during each system boot-up.

Figure 3 shows only the basic principle of the system boot-up-process with some simplifications.

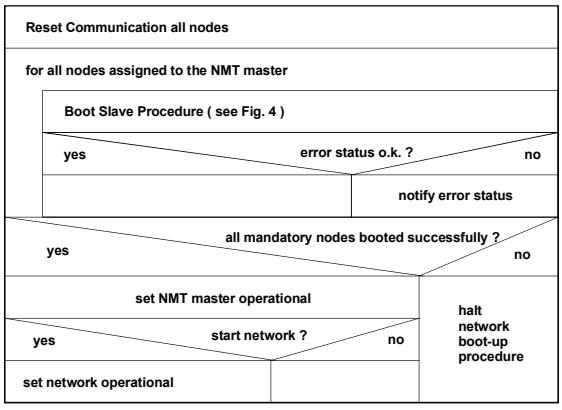

Figure 3: CANopen system boot-up procedure (only basic principle)

First the NMT-Master performs a reset of the communication interface of any of the assigned nodes to ensure a properly defined environment. Then, any of the assigned slave devices are booted individually. The basic principle of this procedure is shown in Figure 4.

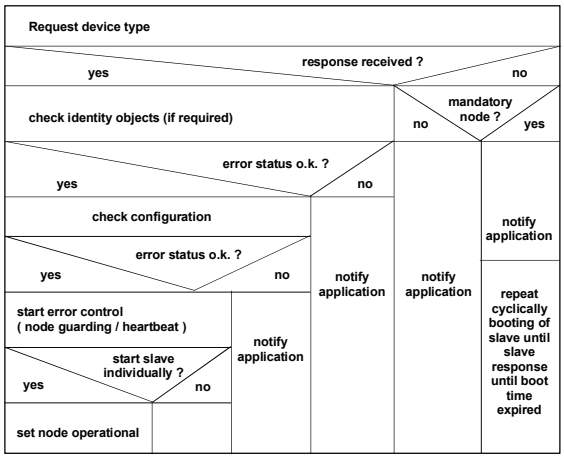

Figure 4: CANopen Slave boot-up procedure (principle)

To boot a Slave device, the NMT-Master checks the Device Type by reading of the “Device-Type”- object (1000H) of the device and compares it with the device type stored in the corresponding network identification object. If there is no response to the Device Type read request and the device is mandatory, the application process is notified and the system boot-up process stopped after expiration of the “boot-up”-time. If on the other hand a non-mandatory device does not answer a read Device Type request, the read request is repeated until the slave is found or the application process stops polling. This allows the system to identify even devices which are later connected to the network.
Of course, if the device type of a node does not correspond to the stored device type, the application is notified and the Slave boot-up procedure stopped.

After a successful verification of the device type further identification data of the device is checked, in case this data is provided by the device.

Optionally (not shown in Figure 4) the Slave boot-up process continues with verifying the version of the application software including an automatic update of the application software in case of a ver-

sion inconsistency. This function is performed in cooperation with the Configuration Manager (CM). The function of the CM is described later.

After the successful verification of the identification data, the device configuration date and time parameters are being read by the CM. If the CM detects a mismatch of this data with the corresponding values stored by the CM, or if the CM is requested to configure the device with each boot-up, it initiates downloading of the configuration data to the device.

After a successful verification or downloading of the device configuration data the Slave error control services are finally started by the NMT-Master. In case that the device refers to the Heartbeat protocol, the reception of a Heartbeat message within the NMT-Masters Heartbeat Consumer Time is being checked. If the device supports Node Guarding, it will be started right after.

When all nodes are successfully booted (see Figure 3) the NMT-Master sets itself and all its assigned Slaves into the "operational" state. Now the transmission of process data via PDOs is enabled.

The Configuration Manager

The main function of the Configuration Manager (CM) is to configure a device during boot-up if requested. Therefore the CM uses the configuration data provided by the Device Configuration File (DCF) of the device. If the CM is not located on the computer used for setting-up the DCF, it is necessary to transfer the DCF via CANopen. The DCFs of the network devices are written to the CM, e.g. by a configuration tool via the "Store DCF" – object (1F20H) with the sub-indices corresponding to the node-ID of the devices. The DCF of a device can be read from the CM by reading the "Store DCF" – object with sub-index according to the node-ID of the device. A DCF can be stored "as it is" or in compressed form.

To further reduce the storage requirements a concise configuration data format is specified in DSP 302. This format does not contain all of the information included in the DCF. The information to be stored consists of the parameter values of the object dictionary entries which differ from the default values and is written to the CM in form of a stream of data to the "Concise DCF" – object (1F22H) with the sub-index equal to the node-ID of a device.

To allow the CM to determine whether a device needs to be reconfigured, DS 301 provides the "Verify Configuration" – object (1020H). If a device supports the saving of parameters in non-volatile memory, the CM uses this object to verify the configuration after a device resets and to determine if a reconfiguration of the device is necessary. Therefore the configuration tool has to write date and time of the configuration in this object.

To check the validity of the configuration data of a device after a reset, the CM reads the "Verify Configuration" – object of the device and compares the values of this object (date and time of device configuration) with the expected values for date and time, stored in the CM objects "ExpectedConfigurationDate" (1F26H) and "ExpectedConfigurationTime" (1F27H) by the Configuration tool. If the expected values are zero or not identical with the values of the "Verify Configuration"- object, the CM starts downloading the configuration data from the DCF of the device.

3. Configuration of Distributed Systems

Based on the concept of the object dictionary and a complete free choice of type and structure of communication relationships between application objects, CANopen provides a very high degree of flexibility with respect to device and system configuration.

On the other hand, this high flexibility makes configuration of a CANopen system a rather complex matter if this is not supported by a configuration tool. This is especially true, for the configuration of distributed intelligent systems with more than one interacting programmable device and a complex communication structure. A

more detailed discussion of this topic is given in [3].

3.1 The Configuration Process

The main steps for the configuration of a CANopen-based automation system are:

- Definition of the network configuration: Selection of the devices on the network, allocation of node-Ids and baudrate.

- Configuration of the specific device functionality. For programmable devices this includes the definition of the required network variables.

- Set-up of the communication relationships between all application objects shared across the network and of the required service data channels between devices.

Figure 5 illustrates the basic task of setting-up the communication relationships for transmission of process data (application objects) between two intelligent devices. This process includes the following steps:

- Definition of the required input and output network variables according to the application programs running on the programmable devices.

- Mapping of the producer application objects (input network variables and/or local process inputs) of both devices into appropriate Transmit PDOs (TxPDOs). Generally, several application objects can be mapped into one TxPDO. Also the transmission mode (e.g. asynchronous, synchronous) of the TxPDO has to be selected.

- Mapping of the consumer application objects (output network variables or local process outputs) of both devices into appropriate Receive PDOs (RxPDOs). Of course, there can be more than just one consuming device. In that case any of the other consuming devices also need to be configured

- Allocation of appropriate CAN-Identifiers to each TxPDO and the corresponding RxPDO

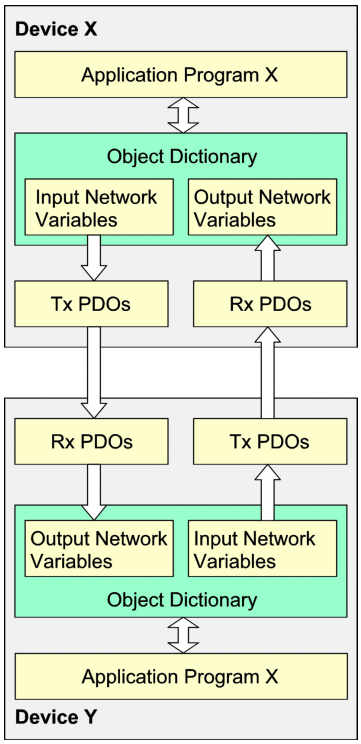

Figure 5: Mapping of network variables into PDOs and PDO-Linking

The described basic process of configuring the transmission of process data shows, that configuring a distributed intelligent CANopen system without a configuration tool is not very efficient. The next section will introduce an existing CANopen configuration tool.

3.2 System Configuration by means of a CANopen Configuration Tool

With the PC-based **CANopen ConfigurationStudio** offered by IXXAT Automation [7] a powerful configuration tool for the configuration of intelligent CANopen systems is available. When the system is combined with a programming system, e.g. for PLC programming, not only configuration but also programming of distributed intelligent CANopen systems is possible. Based on an efficient data base system, the consistent storage of Elec-

tronic Data Sheets, as well as further, project-specific information is ensured.

Via the tool's **Control Panel** the basic administration of several projects is provided. Further functions of the Control Panel are: Selection and control of the different tools, structuring and definition of networks, handling of Electronic Data Sheets and Device Configuration Files, allocation of node-Ids to network devices, definition of the baud rate as well as utility functions like Windows and file handling options.

The following plug-ins are available:

- **Object Dictionary Browser**

With this tool browsing of the CANopen object dictionary of a device is possible and object dictionary entries can be easily edited.

- **Device Configurator**

This tool specifically supports editing a device's communication and application parameters, PDO-mapping and PDO linking.

- **Visual Object Linker**

Whereas the usage of the Object Dictionary Browser and the Device Configurator is mainly of interest for someone familiar with CANopen (like developers), the Visual Object Linker tool (Figure 6) very efficiently supports the configuration of communication relationships between process data and/or network variables on the application level. Therefore this tool requires almost no knowledge of CANopen. With the Visual Linker it is only necessary to link the producer application object (like a specific Output Network Variable) to one or more consuming application objects. Mapping of the application objects into a TxPDO, allocation of CAN identifiers to Transmit and Receive PDO as well as demapping on the consumer side is performed automatically by the Object Linker. Therefore, the CANopen communication mechanism are completely hidden. The Object Linker also supports the definition of network variables and the export of specified variables to an integrated or external programming system.

- **Network Access Interface**

This plug-in builds the interface to a CANopen network and provides functions for down- and uploading of configuration and program data, scanning of a network, verification of device configurations, network and program control as well as Layer Setting Services.

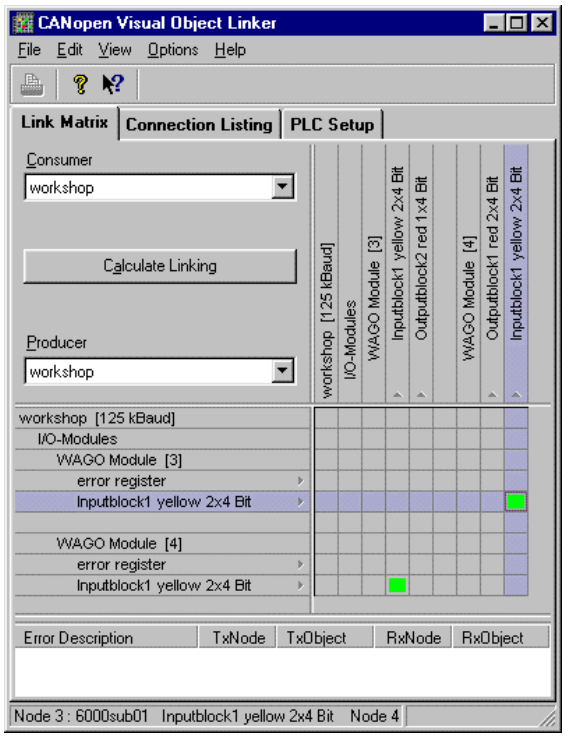

Figure 6: IXXAT CANopen ConfigurationStudio, Object Linker plug-In.

4. Implementation of Programmable CANopen Devices

Since "intelligent" functions in a distributed CANopen system are located mainly on programmable devices like PLCs or HMI devices, these devices should principally be able to perform the function of a CANopen Master as well as that of a CANopen Slave. Based on the CANopen Master software package of IXXAT Automation the implementation of a CANopen controller device is considerably facilitated. In addition to the basic functions supported by standard CANopen software packages, the IXXAT CANopen Master software package (Figure 7) includes all the additional functions necessary for the control of distributed intelligent systems.

The included Network Manager provides a configurable, standardized system boot-up procedure, the Configuration Manager the configuration functionality as defined in CiA DSP 302.

The CANopen Kernel comes with a universal application programming interface for integration with the PLC run time system and the application program, respectively, consisting of a “Process Image” as the interface for process data, as well as a command and a diagnosis interface (Figure 7).

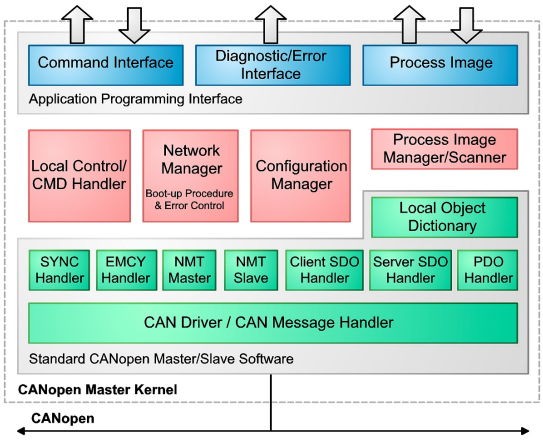

Figure 7: Software architecture of the IXXAT CANopen Master software package

The command interface actually consists of two command buffers: One for local commands and one for commands transmitted by means of SDOs. Via the local command buffer the application program can control the different functions of the CANopen Master software package. Typical commands of this type are, for example, requesting the storage of configuration data, starting the boot-up process or requesting information about the automatically configured Process Images.

Through the remote command buffer commands the reading and writing of the object dictionary entries of the local and remote nodes is performed.
Each command returns a confirmation value which indicates the successful or non-successful execution of the command in the corresponding confirmation buffer.

The diagnosis and error Interface serves to inform the application program about the error statuses of the Master software package, the communication system and the nodes. This interface also provides node emergency messages and error statistics as well as the nodes configuration status and many further status data.

Via the Process Images Input and Output, the application program exchanges process data with the CANopen kernel, which is responsible of transferring the process data between the process images and the remote nodes. The process images are implemented in form of a global data structure accessible by the PLC run time system and the Master software package.

5. Summary

With CANopen DSP 302 a versatile framework for the implementation of distributed intelligent automation systems based on CANopen is available. This includes sophisticated features like network variables, standardized system boot-up, automatic reconfiguration and program download and control. In addition, efficient system configuration and programming tools are available in the market as well as powerful software packages for the implementation of programmable CANopen devices. Therefore, due to the many benefits of distributed intelligent systems, it is expected that CANopen-based automation systems with distributed intelligence will become more and more popular in the near future.

References

[1] http://www.mauell.com

[2] CiA Draft Standard 301, CANopen – Communication Profile for Industrial Systems, Version 4.01, 6/2000

[3] Etschberger, K.: Controller Area Network, Basics, Protocols, Chips and Applications. IXXAT Press, 2001. ISBN3-00-007376-0

[4] CiA Draft Standard Proposal 302: Framework for Programmable CANopen Devices, Version 3.0, 6/2000. CAN-in-Automation

[5] CiA, Draft Standard Proposal 405: Interface and Device Profile for IEC 61131-3 Programmable Devices. Version 2.0, 2/2000

[6] Etschberger, K., Eberle, J.: CANopen becoming intelligent with IEC 1131-3. CAN in Automation. Proceedings of the 5th international CAN Conference. 11/98.

[7] http://www.ixxat.de

Prof. Dr.-Ing. K. Etschberger
IXXAT Automation GmbH
Leibnizstr. 15
D-88250 Weingarten
e-mail: etschberger@ixxat.de
Phone: +49-0751-56146-0
Fax: +49-0751-56146-29
web: www.ixxat.de

Dipl.-Ing. Ch. Schlegel
IXXAT Automation GmbH
Leibnizstr. 15
D-88250 Weingarten
e-mail: schlegel@ixxat.de
Phone: +49-0751-56146-0
Fax: +49-0751-56146-29
web: www.ixxat.de