Towards Autoconfiguration of Industrial Automation Systems: A Case Study Using Profinet IO

Lars Dürkop[1], Henning Trsek[1], Jürgen Jasperneite[1,2], and Lukasz Wisniewski[1]

[1]inIT - Institute Industrial IT, Ostwestfalen-Lippe University of Applied Sciences, D-32657 Lemgo, Germany {lars.duerkop, henning.trsek, juergen.jasperneite, lukasz.wisniewski}@hs-owl.de

[2]Fraunhofer IOSB-INA Application Center Industrial Automation, D-32657 Lemgo, Germany juergen.jasperneite@iosb-ina.fraunhofer.de

Abstract

Nowadays the deployment and commissioning of complex production processes or Internet-enabled applications interacting with production systems requires a time consuming and error-prone manual system configuration process. This is due to the need to maintain a high level of determinism, safety, and security of the production process itself and avoiding both safety-critical failures and costly production interruptions. The project "IoT@Work - Internet of Things at Work" aims at delivering tools and runtime mechanisms to significantly simplify commissioning, operating, and maintaining complex production processes, thus enabling Plug-and-Produce for automation systems. This paper presents a novel approach for the autoconfiguration of real-time Ethernet systems, and all relevant mechanisms. A case study, based on Profinet IO, evaluates the approach and shows its feasibility.

1. Introduction

Shorter product life-cycles, changing market conditions, and the trend to mass customization are posing several challenges for today's manufacturing companies. They must be able to react in an adaptive manner to produce individualized products requested by their customers. To meet these requirements production processes must be frequently changed and as adaptable and agile as possible.

However, modifications of current production systems require several steps which are time consuming and have to be done manually. Furthermore, they are critical, because failures can lead to very costly production interruptions (e. g., automotive companies produce a car each minute).

The main focus of the project IoT@Work [1] will be the development of technologies required to enable Internet of Things (IoT)-based applications and processes in the manufacturing domain. Thus providing a significant contribution to an Internet of machines, e. g., for realizing such adaptable manufacturing systems. The extension of already existing mechanisms and technologies to enable a secure Plug-and-Produce is the main part of the research. For instance, devices have to be autoconfigured and ready to co-operate with each other as soon as they are plugged into the factory network. This allows a self-adaptation to changes in response to demands, faults, etc.

The contribution of this paper is a new concept for the autoconfiguration of real-time Ethernets (RTE) based on a configuration service which is available throughout the network and an ad-hoc communication channel available within every device. This ad-hoc communication channel will be used to configure the device and supply all relevant parameters before the real-time communication is actually started. A case study evaluates and shows the feasibility of the approach.

The remainder of this paper is organized as follows. After this short introduction, the scope of this work is highlighted in section 2. In section 3 relevant background is discussed, followed by an analysis of the current mechanisms used by Profinet IO (PN IO) in section 4. After this a new concept for autoconfiguration of PN IO is described in section 5 and evaluated by means of a case study in section 6. The paper is concluded in section 7 and an outlook about future work is provided.

2. Scope of this paper

The vision of autoconfiguration in the field of automation can be described with the term Plug-and-Produce

[1]This research work was financially supported by the EU Project IoT@Work under grant number ICT-257367.

© 2012 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. DOI: http://dx.doi.org/10.1109/ETFA.2012.6489654

(PnP) [1]. In an ideal case all devices of a production process can be plugged together and the whole process can start without manual configuration efforts. Without considering physical aspects the essential requirements to enable PnP can be derived from the scenario of a modular production process.

A modular production process consists of several individual plant modules. Each module is equipped with a separate programmable logic controller (PLC) being programmed with its corresponding control software. The software needs specific information from other field devices (e. g. PLCs, robots or drives) depending on their functionality. This data is commonly referred to as process data which has to be exchanged in real-time to meet the requirements of the process. The engineering phase for each of the involved devices can be divided into three manual steps, usually done offline and prior to setting up the system:

i) An engineering tool is used to implement the control software for a certain process, e. g. by using IEC 61131 tasks implemented in a PLC.

ii) The physical structure of the automation system, including all devices, is added to the engineering tool and configured in it.

iii) The logical variables of the control software are mapped to the corresponding physical data of the field devices.

Afterwards the engineering tool compiles the provided information and transfers it to the devices. In order to autoconfigure such a system three essential requirements can be identified:

1. Autoconfiguration of devices Plug-and-Produce requires the ability to add and remove devices from the network without manual configuration of them.

2. Identification of required signals The assignment of process data to the variables of the control software has to proceed without manual interventions. A solution to this problem could be the definition of the semantic of all signals in a machine-readable and function-describing way. This is part of current research activities and not in the scope of this paper.

3. Real-time communication In the automation industry networks on the shop floor based on RTEs are becoming state of the art. RTEs need to be configured during the design of the system in an engineering tool.

In this paper a Plug-and-Produce capable system is proposed which is able to autoconfigure industrial devices and the corresponding RTE. Therefore, an ad-hoc communication channel is introduced which co-exists with the real-time channel. The ad-hoc channel allows the devices to exchange information, e. g., to discover other existing devices, without manual configuration needs. The ad-hoc channel is implemented by using a Service-Oriented Architecture (SOA) which offers the requested capabilities. The use of SOA is also important for the extension of the proposed approach in the future. In the field of factory automation, SOA allows to build an agile and flexible automation system able to be adapted easily to changes [2]. In the proposed solution the ad-hoc channel is used for the autoconfiguration of the RTE PN IO. Therefore, the manual configuration steps of PN IO are described and a concept for RTE autoconfiguration is derived based on this description.

3. State of the art

In the field of residential and office environments autoconfiguration of new devices is commonly known under the term Plug-and-Play. Most prevalent standards are Universal Plug and Play (UPnP) [3] and the Universal Serial Bus (USB) [4]. The latter is a standard for connecting peripheral devices to a host, e. g. a storage device to a PC. UPnP provides methods for automatic integration of devices in an IP based network without configuration and setup by means of specifying services for discovery, description and control of devices. Hence, it is an approach to realize an SOA on the device level. However, UPnP is not suitable in factory automation environments, since it uses broadcast messages for discovering and controlling. In a network with a huge amount of devices this might congest the network and overload the resource constraint devices. Furthermore, UPnP is limited to only one broadcast domain [5], [6].

The SIRENA (Service Infrastructure for Real-time Embedded Networked Applications) project [7] developed a SOA framework for resource constraint devices which are frequently deployed in industrial automation and other domains. In [6] several technologies were investigated regarding their suitability. As a result the Devices Profile for Web Services (DPWS) [8] has been chosen as base technology for SIRENA. DPWS messages are transported with the XML-based SOAP protocol [9], searching and locating devices is done by the multicast protocol WS-Discovery. A full DPWS documentation can be found under [10]. The services offered by DPWS are similar to these offered by UPnP and it can be considered as a potential successor of UPnP [2]. Open source implementations of DPWS are available in the projects WS4D [11] and SOA4D [12]. Since the DPWS approach lacks real-time capabilities, [13] tries to improve the performance of DPWS by a binary encoding of SOAP messages which reduces their size to 20%. In [14] some parts of the DPWS stack are adapted to a real-time operating system. This allows an estimation of the maximum time needed for processing and transferring data of a certain service.

All previously described solutions use Ethernet as a basic communication technology. This leads to the fundamental problem that they cannot support real-time communication since standard Ethernet does not provide this feature. To address this issue, real-time capable network technologies, such as RTEs, must be used. They require a manual and time consuming offline configuration which is opposed to the adaptability of SOA and the vision of PnP.

The PAPAS Project [15] introduced a middleware between an RTE and the application layer. The middleware offers a uniform access to different RTEs. Field devices can be plugged and unplugged during operation of the system. In order to work properly, all devices and software must implement the PAPAS architecture and the PAPAS communication stack. Unfortunately the PAPAS documentation does not explicitly explain how an RTE is reconfigured during its operation when a new device is added or removed.

[16] presents a method to discover the topology of an RTE automatically and to allocate local MAC addresses to devices depending on their location in the network. This results in an automatic layer 2 address allocation, but the remaining steps still have to be configured manually.

A promising approach is the autoconfiguration mechanism for the RTE Powerlink, described in [17]. It analyzes the configuration process of an RTE and divides it in five abstract steps. The autoconfiguration solution presented in this paper adopts parts of that methodology, but is more general and the specific steps are quite different.

Table 1. Comparison of different autoconfiguration approaches

	Identification of signals	Autoconfiguration	Real-time support	SOA
UPnP [3]	–	⊕	–	⊕
DPWS [10]	–	⊕	–	⊕
PAPAS [15]	–	–	⊕	⊕
Autoconf. of RTEs [16]	–	⊕	⊕	–
Powerlink autoconf. [17]	–	⊕	⊕	–

In Table 1 the features of the different existing solutions are summarized with respect to the requirements identified in section 2. To sum it up, there are two open research questions for enabling PnP in real-time automation environment using an SOA. First, to the best of the authors knowledge, no SOA is currently available in combination with an RTE. Second, the use of an RTE requires always manual configuration of the network. The proposed approach shows a way how the configuration of the RTE PN IO can be automatized. Furthermore the DPWS stack is implemented in the devices of our solution. Its methods are part of the autoconfiguration process and in applications based on our work DPWS can be used to built a SOA where the critcial process data can be exchanged over the RTE.

4. Description of Profinet IO

PN IO is a real-time Ethernet based on IEEE 802.3 (Fast Ethernet) designed for real-time (RT) communication between a Controller (typical a PLC) and IO-Devices. It allows the use of standard TCP/IP applications and the simultaneous exchange of real-time process data over the same network. RT frames are embedded directly in an Ethernet frame without using any other higher layer protocols like UDP or TCP, i. e. they bypass the TCP/IP stack and can be handled directly by the PN stack of a device.

The standard channel is used for the context management. It handles configuration and diagnosis of PN devices. Cyclic process data, events and alarms are sent over the RT channels. PN defines different classes of RT communication. In real-time class 1, RT data is sent over an unsynchronized RT channel. RT capabilities are ensured by prioritization of RT frames within the Ethernet switches by using VLAN tags. It demands switches with the IEEE802.1Q protocol support. Send cycle times of less than 10 ms are possible. In RT class 3, the isochronous RT (IRT) mode, all PN devices and switches are synchronized to one clock and the IRT data is sent in pre-configured and scheduled timeslots. Send cycle times down to 31.25 µs can be achieved in this mode. The usage of this mode requires special IRT hardware. The autoconfiguration of IRT communication will be part of our future work.

4.1. Basic Profinet IO characteristics

A PN device is using a slot/sub-slot device model at the application layer. A slot represents physical or logical modules of a device. It is divided into subslots to which IO data, alarms and diagnostic data of a slot are assigned. The subslots are the interfaces to the process. They define also the format of the data. The different modules of a device and their segmentation into submodules are described with the xml-based Generic Station Description Markup Language (GSDML) in a device specific Generic Station Description (GSD) file. In addition, several user profiles can be defined per device. Every user profile is addressed by an Application Process Instance (API). The API defines the slots and subslots used by the profile.

PN IO defines three roles of devices: IO-Controller, IO-Supervisor and IO-Device. The IO-Devices are the peripheral field devices which form the interface to the automation process. IO-Devices can be divided in two types. A block IO-Device has a fixed module configuration described in the GSD file, whereas a modular IO-Device allows to add and remove modules depending on the requirements. The GSD file contains all possible modules. The exact configuration of the IO-Device must be defined in the engineering tool. Our solution handles the autoconfiguration of block IO-Devices only. [18] shows a way how the actual module configuration of modular IO-Devices can be read out by the Simple Network Management Protocol (SNMP). This solution could be integrated in future versions of our autoconfiguration concept to enable the usage of modular devices.

The IO-Controller is typically part of the PLC and controls the automation process. The process data between the IO-Devices and the IO-Controller are exchanged in real-time and in pre-defined cyclic time intervals. An IO-Supervisor is a device for programming the IO-Controller or an HMI. It can also monitor the PN traffic.

Between the IO-Controller and every IO-Device one Application Relation (AR) is established. Within an AR the data to be exchanged is defined in different Communication Relations (CR). Cyclic process data is using the IO data CR, the configuration and other acyclic data is exchanged in a Record Data CR and real-time alarms in an Alarm CR.

4.2. Profinet IO start-up phase

After powering on the PN IO system the start-up phase is immediately initiated. In this phase the IO-Controller must configure all IO-Devices. When the start-up phase ends successfully the cyclic process data exchange starts. The IO-Controller receives the necessary configuration data from an engineering tool. This section describes the start-up phase in detail and is used to derive necessary modifications of it, to realize the PN IO autoconfiguration presented in section 5.

Before a PN system can start its operation, every PN device needs a device name which is assigned by the engineering tool and is known to the IO-Controller. In the first step the IO-Controller assigns an IP address to every IO-Device by the use of the Discovery and Configuration Protocol (DCP).

When the assignment of the IP addresses is completed, the start-up procedure continues with the messages shown in Figure 1.

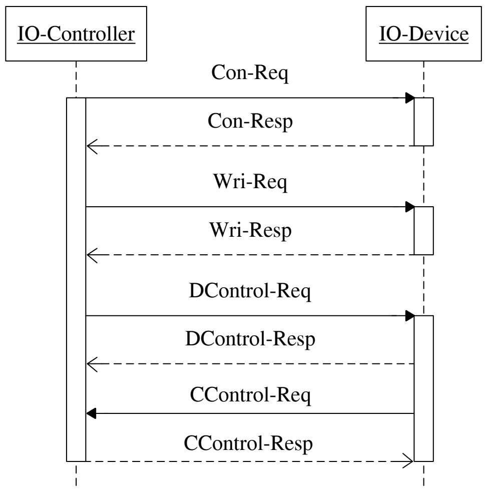


Figure 1. Start-up procedure

The IO-Controller establishes the AR and the corresponding CRs to every IO-Device with a Connect-Request (Con-Req). The Con-Req consists of several blocks whose descriptions and most important properties are listed below:

- AR block: contains a unique identifier (ARUUID) for the AR.
- Input CR block: defines a CR for the cyclic data from the IO-Device to the IO-Controller. The parameters are among others the used real-time class, a frame ID and timing of the data. Furthermore this block contains the definition of the APIs. The APIs describe which slots and subslots are transferred to the IO-Controller.
- Output CR block: same as above for the data from the IO-Controller to the IO-Device.
- Expected submodule block: this block links the slots/subslots from the input and output CR blocks to the modules/submodules of the device as specified in the GSD file.
- Multicast Communication Relation (MCR) block: contains data to establish a data communication between several IO-Devices. This is not further considered here.
- Alarm CR block: specifies priorities of alarms and the maximal length of an alarm.

The IO-Devices acknowledges the Con-Req with a Connect-Response (Con-Resp). In case of no errors, the Con-Resp mirrors the configuration data from the Con-Req. If not all expected modules are present in the IO-Device it answers with a Module Diff block.

Submodules could need additional parameterization data which are specified in the GSD file. The IO-Controller sends one Write-Request (Wri-Req) to each IO-Device which contains these data. The IO-Device confirms the reception with a Write-Response (Wri-Resp).

The end of the parameterization process is signaled by the DControl-Request from the IO-Controller and the corresponding response. When the IO-Device has processed the received data, it signals its operational readiness with the CControl-Request. The start-up procedure ends with the acknowledgement in a CControl-Response of the IO-Controller.

5. Autoconfiguration concept using Profinet IO

In conventional operations the IO-Controller obtains its configuration data from the engineering tool. In a PnP scenario this data is not available. The solution presented in this paper is based on the extension of PN devices by an ad-hoc communication channel and a configuration service implemented in the IO-Controller or an distinct PnP server. The ad-hoc channel can be used with all RTEs which allow standard TCP/IP communication without manual configuration. Using this channel the parameters needed for the PN configuration are exchanged. Figure 2 illustrates our approach with the necessary extensions to enable PnP functionality.

For a complete PnP-enabled solution the variables of the application (e. g. an IEC 61131 based PLC) have to

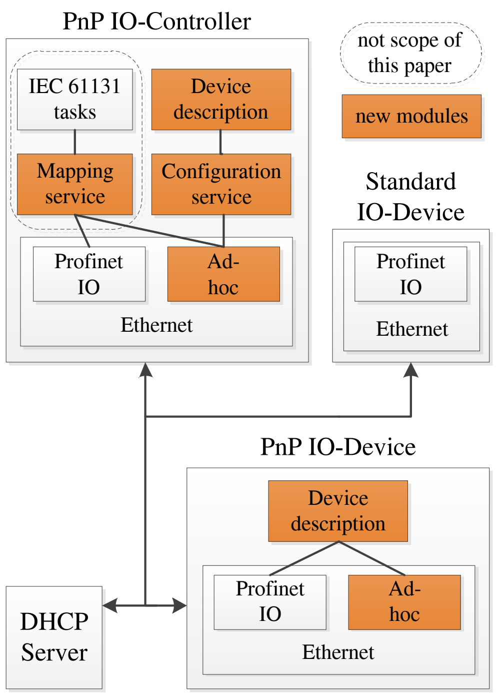


Figure 2. Autoconfiguration scenario

be mapped to the process data of the IO-Devices as mentioned in section 2. Here, the focus is on the autoconfiguration of the PN stack. It means that an IO-Device can be attached to the network and the process data exchange between the IO-Device and the IO-Controller starts without any manual intervention and pre-configuring of the IO-Controller. The IO-Device can be added at any time of the operational time of the network.

The process of autoconfiguration can be divided in five steps according to [17]:

1. Physical connection establishment: The device must be connected to the network, meaning that an existing network infrastructure (e. g. switches which support VLAN tagging and packet prioritization) must be available. It is not necessary to take the network topology into account, because PN supports various topologies, such as star, tree, line, ring and various combinations of them.

2. Discovery: The newly connected device must inform the IO-Controller about its presence or the IO-Controller has to scan the network if there are any new devices.

3. Basic Communication: The IO-Controller needs an information about the new device. For this purpose an ad-hoc communication channel to the IO-Device is needed to download the GSD file. Another possibility is to download the GSD file from a central file repository.

4. Capability Assessment: From the retrieved device description file the configuration service of the IO-Controller has to determine all necessary parameters for the PN stack.

5. Configuration: The PN stack has to be configured according the results obtained in the step 4. Afterwards the cyclic process data exchange between the IO-Controller and the new device can start.

In the following subsection the abovementioned steps are described for the proposed approach for two different cases, for PnP-enabled IO-Devices and for standard IO-Devices.

5.1. Autoconfiguration with PnP-enabled IO-Devices

For the steps 2 and 3 of the autoconfiguration process the ad-hoc communication channel of our concept is used. If only the requirements discovering and transferring of the GSD file were considered, the use of standards coming from the TCP/IP family are sufficient. As stated in section 3, a trend to use SOA in factory automation can be observed. To easily integrate PnP-enabled PN devices in a SOA environment in the future, we decided to realize the ad-hoc channel with Web Services. DPWS is used because of its suitability for embedded devices. Hence, the autoconfiguration can be described as follows.

1. Physical Connection: The new device can be attached to the network using any free port of an Ethernet switch. No special actions have to be taken before. This complies with the complete Plug-and-Play definition from [15].

2. Discovery: Since the new device must be able to receive an IP address from an existing DHCP server within its network segment, a DHCP client is added to the IO-Device (usually not available on standard IO-Devices). The discovery process uses the WS-Discovery and WS-Transfer protocols [8] provided by the DPWS stack. Both protocols enable devices to discover specific services in a network and to exchange device metadata. Whenever a new IO-Device is connected to the network, it sends a hello message to the WS-Discovery multicast IP address which is defined in the DPWS standard. The IO-Controller recognizes the new device and retrieves its metadata (e. g. manufacturer and device name, firmware) using the WS-Transfer protocol.

3. Basic Communication: The IO-Device hosts its own GSD file. The configuration service on the IO-Controller can retrieve this file by invoking the Get-GSD service of the IO-Device. This service transfers the GSD-file via the SOAP Message Transmission Optimization Mechanism (MTOM) [19].

4. Capability Assessment: The configuration service parses the received GSD file. Based on this data it has to define the parameters needed to establish the cyclic process data exchange. First the configuration

service must define a PN device name. It is based on the device identifier string contained in the GSD file. If a device with the same identifier exists already in the network, a consecutive number is added to the device name. After this the device name is sent to the IO-Device by a DCP Set-Request. The other parameters are used in the Connect-Request to establish an AR to the IO-Device. Referring to the description in section 4.2 these parameters are:

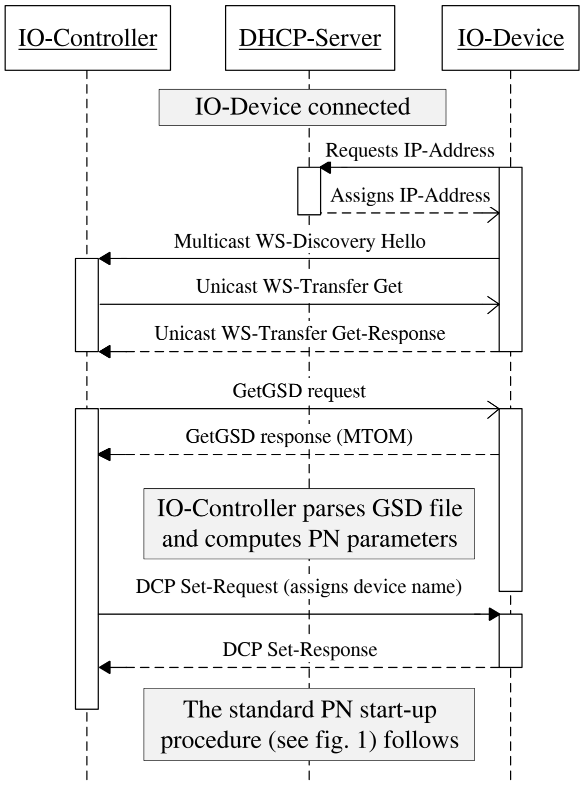
- AR-Block: the ARUUID can be defined randomly, but must be unique.
- Input CR block: One CR for the cyclic data exchange from the IO-Device to the IO-Controller is defined. RT class 1 is defined as standard value. The classes 2 and 3 are used by PN IRT and are not yet supported by our approach. The frame ID is chosen from the predefined range which PN specifies for RT class 1. The basis for the calculation of the timing parameters is the *MinDeviceInterval* considered in the GSD file. It states the minimum time between two send cycles of the IO-Device. If the *MinDeviceInterval* value is larger than a default value (i. e. 8 ms), the send cycle is set to the *MinDeviceInterval* instead of the default value. One API is configured which includes all input signals of the IO-Device. The inputs are extracted from the GSD file.
- Output CR block: One CR for the cyclic data exchange from the IO-Controller to the IO-Device is defined. Besides the data for the outputs of the IO-Device this block contains the acknowledgements for the data of the input CR.
- Expected submodule block: Here, all submodules from the GSD file are instantiated. The MCR and alarm CR block are not necessary to start the cyclic process data exchange and are not used yet. Some submodules could need additional parameterization data from the GSD file. In this case a Write-Request frame containing this data is composed.

5. Configuration: The configuration service sets all necessary parameters of the PN stack of the IO-Controller.

Figure 3 shows a sequence diagram of the autoconfiguration process.

5.2. Autoconfiguration with standard IO-Devices

The main restriction in the autoconfiguration process of a standard IO-Device is the fact that the IO-Device cannot send its GSD file to the IO-Controller. This would require manual adaptions of the firmware which is impossible in most cases. Hence, the files of legacy IO-devices must be stored in a central repository reachable within the network. This could be the IO-Controller itself or a dedicated FTP-Server. The autoconfiguration of standard IO-Devices varies from the autoconfiguration of PnP-enabled devices in the following points:



IO-Controller
DHCP-Server
IO-Device
IO-Device connected
Requests IP-Address
Assigns IP-Address
Multicast WS-Discovery Hello
Unicast WS-Transfer Get
Unicast WS-Transfer Get-Response
GetGSD request
GetGSD response (MTOM)
IO-Controller parses GSD file and computes PN parameters
DCP Set-Request (assigns device name)
DCP Set-Response
The standard PN start-up procedure (see fig. 1) follows

Figure 3. Autoconfiguration of a PnP-enabled IO-Device

- Discovery: For the standard IO-Device the DPWS stack cannot be used for discovery, because the firmware can not be modified with additional functionalities. Furthermore common IO-Devices do not include a DHCP client. Instead, the IO-Controller can use the DCP protocol to discover new IO-Devices. Therefore the IO-Controller sends regularly Identify-Requests with the ”all”-option to the PN multicast MAC address. All IO-Devices answer with Identify-Responses containing their PN device name (if available), MAC address and a device identifier. Based on these answers the IO-Controller can recognize new devices. The IO-Device does not receive an IP address yet.
- Basic Communication: The IO-Controller can identify an IO-Device by its device identifier which was submitted in the Identify-Response frame. Based on this identifier the IO-Controller can retrieve the appropriate GSD file from a FTP-Server. The IP address of the FTP-Server is preconfigured and the GSD files of all possible IO-Devices have to be available on the server before they are connected to the network.

- Capability Assessment: This step is similar to that one from section 5.1. The only difference is that the IO-Device has not received an IP address because of the missing DHCP client. Therefore a DHCP-like function is implemented in the IO-Controller. This function chooses an IP address and assigns it (together with the PN device name) to the IO-Device in the DCP Set-Request.

The whole procedure is shown as a sequence chart in Figure 4.

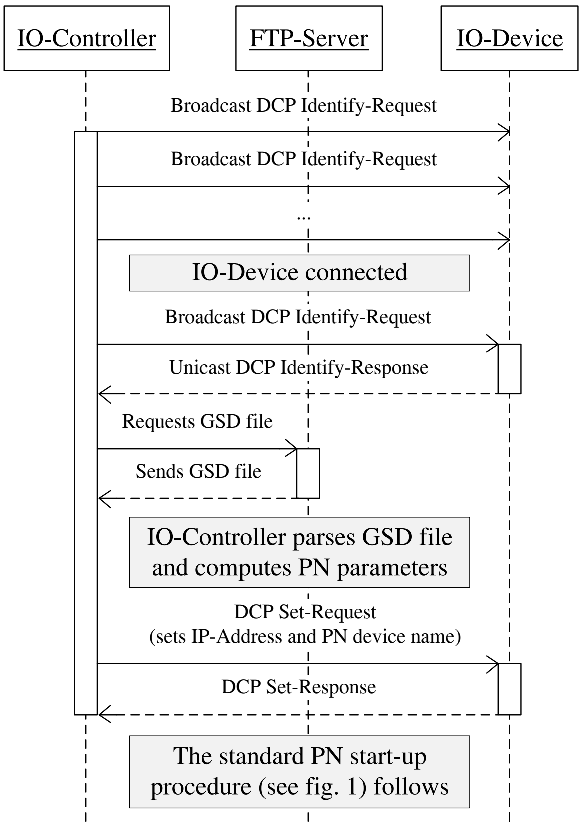

IO-Controller
FTP-Server
IO-Device
Broadcast DCP Identify-Request
Broadcast DCP Identify-Request
...
IO-Device connected
Broadcast DCP Identify-Request
Unicast DCP Identify-Response
Requests GSD file
Sends GSD file
IO-Controller parses GSD file and computes PN parameters
DCP Set-Request (sets IP-Address and PN device name)
DCP Set-Response
The standard PN start-up procedure (see fig. 1) follows

Figure 4. Autoconfiguration of a standard IO-Device

6. Case Study

The proposed autoconfiguration concept has been realized as a case study using the test setup architecture shown in Figure 5. A picture of the real setup is shown in Figure 6. It consists of both a PnP-enabled IO-Device and a standard IO-Device.

The PnP-enabled IO-Device consists of an Altera Stratix IV FPGA with an integrated Nios II processor and the Tiger IP-Core [20]. This IP-Core includes a complete IO-Device stack and a three port switch (two external ports and one internal port) which was developed by inIT and Fraunhofer IOSB-INA. The Nios II software

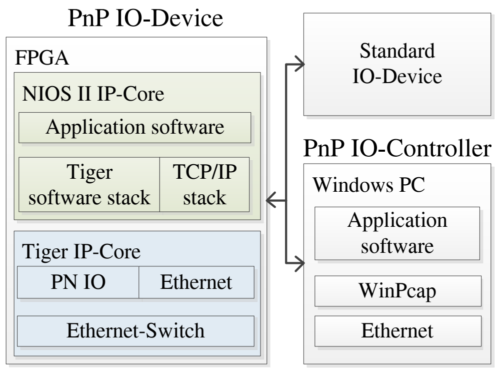


Figure 5. Case study architecture

comprises the real-time operating system embOS [21], a TCP/IP stack and as DPWS package the implementation from the SOA4D (Service-Oriented Architecture for Devices) project [12]. A GSD file for this device was generated and stored locally on the device (within the memory of the FPGA).

As a standard IO-Device we use a Phoenix Contact ILB PN 24 DI16 DIO16 2TX device. It provides 16 digital inputs and 16 digital input/output ports. The GSD file from the manufacturer is saved on an FTP server used for the autoconfiguration of standard IO-Devices.

The IO-Controller is implemented in C++ on a Windows 7 PC based on the WinPcap library [22]. When the IO-Controller detects a new IO-Device by the discovery methods described in chapter 5, it retrieves the GSD file over the DPWS stack or an FTP client. The received file is parsed by the configuration service at the IO-Controller which extracts the information needed for the configuration process. The service composes the necessary PN frames to establish a connection to the IO-Device. The received process data from the IO-Devices is visualized on the screen of the PC.

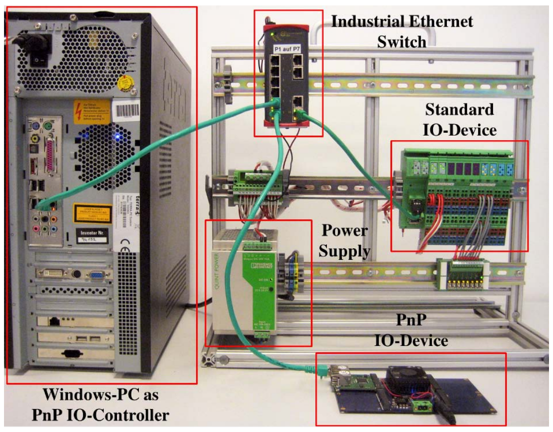


Figure 6. Prototypical case study using PN IO

This setup was successfully tested as follows. The IO-

Devices are reset to their vendor default settings, i. e. they do not have IP addresses or PN device names assigned. After the system is powered on both IO-Devices are connected consecutively to the Ethernet network. When the configuration process has finished, the PC software shows both IO-Devices. The value of each input can be viewed and the values of the outputs can be changed on the PC.

Table 2. Features of the proposed autoconfiguration approach

	Identification of signals	Autoconfiguration	Real-time support	SOA
PN IO autoconf.	*Future work*	⊕	⊕	⊕

Compared with the requirements of section 2 and as shown in Table 2 our solution supports the autoconfiguration of devices, real-time communication for process data exchange and includes an SOA with providing different services, such as the configuration.

7. Conclusion and future work

In this paper a concept for autoconfiguration of industrial automation systems is presented and evaluated by means of a case study using PN IO as an example. The solution was successfully validated on a demonstrator. Currently, PN IO systems must be configured in an offline phase before the system starts up. All devices must be configured in an engineering tool and the configuration must be downloaded to the IO-Controller. With the proposed concept, these manual steps are no longer necessary. The controller of a PN IO network can configure all IO-Devices automatically whenever they are added to the network. Furthermore, devices can be attached or detached from the network during runtime and full operation of the system.

The future work will address the extension of the autoconfiguration capability to modular and IRT PN devices. Since the abovementioned steps are not sufficient to obtain a system being capable of true Plug-and-Produce which involves also the higher layers of the networking stack, we will also address required mechanisms above the network layer. For instance, control software must automatically identify its needed signals from other nodes in the network. This requires the capability of modules to semantically describe their own signals and to negotiate new established connections to other modules.

References

[1] T. Arai, Y. Aiyama, Y. Maeda, M. Sugi, and J. Ota. Agile assembly system by plug and produce. *CIRP Annals Manufacturing Technology*, 49:1 – 4, 2000.

[2] F. Jammes and H. Smith. Service-oriented paradigms in industrial automation. In *IEEE Transactions on Industrial Informatics*, pages 62 – 70, 2005.

[3] Upnp forum homepage. http://www.upnp.org/.

[4] Usb implementers forum homepage. http://www.usb.org/.

[5] B. Oesterdieckhoff, C. Loeser, I. Jahnich, and R. Glaschick. Integrative approach of web services and universal plug and play within an av scenario. In *3rd IEEE International Conference on Industrial Informatics (INDIN)*, pages 123 – 128, 2005.

[6] H. Bohn, A. Bobek, and F. Golatowski. Sirena - service infrastructure for real-time embedded networked devices: A service oriented framework for different domains. In *International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies (ICN/ICON/MCL)*, page 43, 2006.

[7] Sirena project web site. http://www.sirena-itea.org/.

[8] E. Zeeb, A. Bobek, H. Bohn, and F. Golatowski. Service-oriented architectures for embedded systems using devices profile for web services. In *21st International Conference on Advanced Information Networking and Applications Workshops (AINAW)*, pages 956 – 963, 2007.

[9] Soap version 1.2 specification. http://www.w3.org/TR/soap/.

[10] Devices profile for web services version 1.1. http://docs.oasis-open.org/ws-dd/dpws/1.1/os/wsdd-dpws-1.1-spec-os.pdf.

[11] Web services for devices project web site. http://ws4d.e-technik.uni-rostock.de/.

[12] Soa4d project web site. https://forge.soa4d.org/.

[13] F. Jammes. Real time device level service-oriented architectures. In *IEEE International Symposium on Industrial Electronics (ISIE)*, pages 1722 – 1726, 2011.

[14] A. Pohl, H. Krumm, F. Holland, I. Luck, and F.-J. Stewing. Service-orientation and flexible service binding in distributed automation and control systems. In *22nd International Conference on Advanced Information Networking and Applications - Workshops (AINAW)*, pages 1393 – 1398, 2008.

[15] U. Zimmermann, R. Bischoff, G. Grunwald, G. Plank, and D. Reintsema. Communication, configuration, application. the three layer concept for plug-and-produce. In *5th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, 2008.

[16] J. Imtiaz, J. Jasperneite, K. Weber, F.-J. Goetz, and G. Lessmann. A novel method for auto configuration of realtime ethernet networks. In *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 861 – 868, 2008.

[17] G. Reinhart, S. Krug, S. Huttner, Z. Mari, F. Riedelbauch, and M. Schlogel. Automatic configuration (plug & produce) of industrial ethernet networks. In *9th IEEE/IAS International Conference on Industry Applications (INDUSCON)*, pages 1 – 6, 2008.

[18] D. von Rohr, M. Felser, and M. Rentschler. Simplifying the engineering of modular profinet io devices. In *IEEE 16th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1 – 4, 2011.

[19] Soap message transmission optimization mechanism. http://www.w3.org/TR/soap12-mtom/.

[20] Profinet io device chip tps-1. http://www.kw-software.com/com/industrial-ethernet/3021.jsp.

[21] embos product homepage. http://www.segger.com/embos.html.

[22] Winpcap web site. http://www.winpcap.org/.