Using OPC-UA for the Autoconfiguration of Real-time Ethernet Systems

Lars Dürkop[1], Jahanzaib Imtiaz[1], Henning Trsek[1], Lukasz Wisniewski[1], Jürgen Jasperneite[12]
[1]inIT - Institute Industrial IT, Ostwestfalen-Lippe University of Applied Sciences, D-32657 Lemgo, Germany
Email: {lars.duerkop, jahanzaib.imtiaz, henning.trsek, lukasz.wisniewski, juergen.jasperneite}@hs-owl.de
[2]Fraunhofer IOSB-INA Application Center Industrial Automation, D-32657 Lemgo, Germany
Email: juergen.jasperneite@iosb-ina.fraunhofer.de

***Abstract*—In the future, production systems will consist of modular and flexible production components, being able to adapt to completely new manufacturing processes. This requirement arises from market turbulences caused by customer demands, i. e. highly customized goods in smaller production batches, or phenomenon like commercial crisis. In order to achieve adaptable production systems, one of the major challenges is to develop suitable autoconfiguration mechanisms for industrial automation systems. This paper presents a two-step architecture for the autoconfiguration of real-time Ethernet (RTE) systems. As a first step, an RTE-independent device discovery mechanism is introduced. Afterwards, it is shown how the parameters of an RTE can be configured automatically using Profinet IO as an exemplary RTE system. In contrast to the existing approaches, the proposed discovery mechanism is based on the OPC Unified Architecture (OPC-UA). In addition, a procedure to autoconfigure modular IO-Devices is introduced.**

I. Introduction

Flexibility and adaptability are the major challenges for the industrial automation industry [1]. To reach these objectives, one main challenge is to minimize the reconfiguration complexity of the production systems. A typical automation scenario consists of a control software that is executed on a programmable logic controller (PLC). The PLC is connected to the field devices like sensors and actuators forming the interface to the technical process. Nowadays, RTEs are a common solution to allow data exchange between the PLC and the field devices. Most RTEs have in common that they bypass the TCP/IP stack transferring the real-time data. Some also modify the underlying basic Ethernet technology. The needed effort for setting up an RTE system can be divided into the following parts:

1) Software: The control logic of the process must be implemented. This is usually done in a language according to the IEC 61131. The information that the software has to exchange with the field devices (called process data) are defined as an external variables. The software engineer has to map these variables to the corresponding physical I/Os of the field devices.
2) Communication: The parameters for the RTE have to be configured, e. g. cycle times, device names, etc.

To achieve the vision of Plug-and-Produce (PnP), these two parts must be automated. In a PnP system, new devices can be connected to an automation system without any or with very little manual configuration effort. While the software part is still considered to be the future work, this paper will address the communication aspect. It seizes on existing concepts for RTE autoconfiguration and introduces extensions to cover important use cases.

II. Related work

A similar approach of autoconfiguration in IT systems is called Plug-and-Play. The Universal Serial Bus (USB) [2] is a well-known example. USB defines a start-up procedure to identify new devices and uses generic drivers for specific device classes. It is used for connecting peripheral devices to computers. The concept of a standardized identification process is also used in the Service-Oriented Architecture (SOA) concept, mentioned in the following. The generic drivers can be compared with the device descriptions files (DDF) used in several RTEs to describe the RTE specific properties of the devices. Universal Plug and Play (UPnP) [3] is another example used mainly in home and office networks. It defines services for addressing, discovery and control of network devices. Such collections of communication services are called SOA. A potential successor of UPnP is the SOA Device Profile for Webservices (DPWS) [4] that was developed especially for embedded devices. There are several approaches to realize PnP in industrial automation based on DPWS. A state of the art analysis of SOAs in the industrial automation is provided by [5] and [6]. One possible realization of PnP with DWPS is presented in [7]. The authors suggest to extend the PLC's control code by a list of needed devices and their functions. A device manager discovers the network and compares the list of required and available devices. DPWS is used for the device descriptions.

These approaches have in common that they are all based on the TCP/IP protocol. Therefore they do not support real-time communication. The real-time guarantees provided by the RTEs require on the other hand a higher configuration complexity. Only a few solutions specialized on reducing this effort are known to the authors. Nevertheless, RTE autoconfiguration is the basis for the PnP paradigm in automation as stated in chapter I. The authors in [8] describe an algorithm for automatic assignment of MAC addresses to RTE devices based on the network topology. This allows simplified device exchange in case of failures. However, it does not configure the RTE communication parameters. This topic is covered in [9]. The authors introduce a five step model for the RTE autoconfiguration using Ethernet Powerlink (EPL). The new devices are discovered by specific EPL techniques instead

 DOI: http://dx.doi.org/10.1109/INDIN.2013.6622890

of an RTE independent mechanism. This approach is picked up and generalized in [10], where an RTE independent ad hoc communication channel is introduced for discovery and configuration purposes. The ad hoc channel is realized by DPWS, and Profinet IO (PNIO) is used as an RTE. In [11] the suitability of DPWS for resource-constrained field devices is investigated and compared with OPC-UA. It is shown that the resource usage of that special OPC-UA implementation in terms of program memory is considerably lower than in DPWS (10 kBytes in contrast to 1007 kBytes).

In this paper the autoconfiguration procedure proposed in [10] is modified in terms of two major aspects. First, the ad hoc channel is implemented using OPC-UA instead of DPWS, taking the results of [11] into account. Second, the approach presented in [10] covers only PNIO block devices (see section III-B). This significant limitation will be solved by a new method introduced in section IV-B. The remainder of this paper is structured as follows: Section III provides an overview of OPC-UA and PNIO. These are the base technologies of the proposed autoconfiguration approach described in section IV. The validation setup is presented in section V. Finally, a conclusion and outlook about the future work are given in section VI.

III. OPC-UA and Profinet IO – A brief overview

A. OPC Unified Architecture

OPC-UA is a platform-independent industrial middleware technology. It is designed to allow interoperability between heterogeneous system components over various types of networks. OPC-UA defines methods for data modelling and transport. The information is modelled in the OPC-UA address space by using object-oriented techniques [11]. Since the OPC-UA specification provides only an infrastructure to model an information, it does not dictate any particular semantic. It is open to domain specific organizations to define their own respective information models. OPC-UA is not limited to some specific communication protocols. It provides guidelines for the information representation but does not restrict an implementation to a specific data encoding. Furthermore, OPC-UA is envisioned as an enabler for a seamless vertical integration in automation systems and does allow any combination of the client/server component at various levels of the automation hierarchy for a specific application.

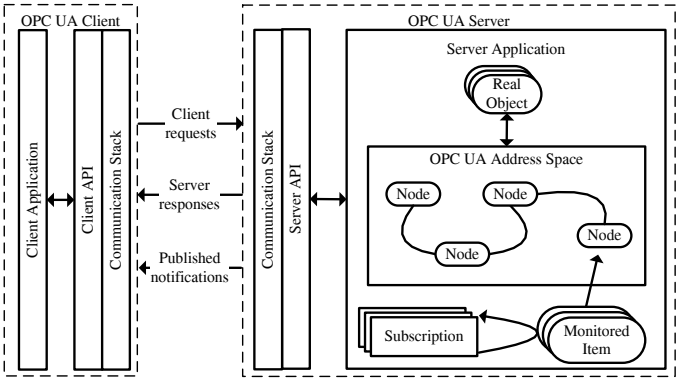

Fig. 1: OPC Unified Architecture

Figure 1 illustrates major elements of a typical OPC-UA client/server architecture and describes how they relate to each other. An OPC-UA client/server application uses an application programming interface (API) to exchange the OPC-UA service requests and responses. The API provides an internal interface isolating the application code from the OPC-UA communication stack. This stack handles transport mechanisms for OPC-UA specific messages. The real objects represent the process application and are accessible via OPC-UA means. Nodes in the address space virtually represent those real objects, their types and references to each other. Furthermore, the address space is a hierarchy of nodes accessible by the OPC-UA clients through a multitude of context specific services. Monitored items are entities in a server created by a client that monitors address space nodes and their real-world counterparts. When they detect a data change or an event occurrence, they generate a notification that is published to a client that has subscribed to that item. Furthermore, clients can control the rate at which a publishing should occur.

B. Profinet IO

PNIO is based on the Fast Ethernet according to the IEEE 802.3 Clause 25 [12] and extends this standard by introducing real-time capabilities. The transmission time of the standard TCP/IP packets over Ethernet is hardly predictable and therefore not convenient for industrial automation applications. PNIO offers different methods to improve the temporal behaviour of frames sent over Ethernet. The real-time communication in PNIO can be divided in three classes, which differ in determinism and hardware requirements. The two most important classes are:

- RT-Class 1. The frames of this class are directly encapsulated into the Ethernet frame. Therefore, they bypass the TCP/IP stack of the involved devices which leads to a lower processing time. Furthermore, frames are marked by the VLAN priority tag [13] and thus prioritized by the switches supporting this feature. The communication is not synchronized, therefore a transmission delay due to queueing may occur.
- RT-Class 3. In this class the transmission time of all frames is precisely planned at the engineering phase. This isochronous real-time (IRT) functionality requires special Ethernet interfaces supporting clock synchronisation between all communication partners.

Besides these real-time classes, PNIO also supports native TCP/IP communication. The autoconfiguration of Profinet's IRT feature is a part of the future work. When the real-time capabilities of PNIO are mentioned in the following, it is always referred to the RT-Class 1.

The devices in PNIO can be divided into three main categories:

- IO-Device. The IO-Devices form an interface to the automation process. They read values from sensors and drive actuators. The values produced and consumed by the IO-Devices are called process data.
- IO-Controller. The IO-Controller controls the automation process and acts as an PLC. Usually one automation process consists of several IO-Devices that

are connected to one IO-Controller. The process data between both device types is exchanged cyclically in real-time.

- IO-Supervisor. IO-Supervisors can also exchange data with IO-Devices. They are used mainly for maintenance and as the human-machine interface.

The data exchange in PNIO is connection-oriented. Before PNIO can start its operation, the user has to download the configuration data from the engineering tool to the IO-Controller. The device names that are used for addressing in PNIO must also be set in advance. Afterwards, the communication is initiated by the IO-Controller. Details of the start-up procedure can be found in [10].

To create a configuration for the IO-Controller the engineering tool must be aware of the available devices and their properties that are stored in the device description files (DDF). In case of PNIO, these files are called Generic Station Description (GSD) and are written in the General Station Description Markup Language (GSDML) that is based on XML. The structure of a GSD file is shown in Fig. 2. Due to its complexity only a partial description has been provided.

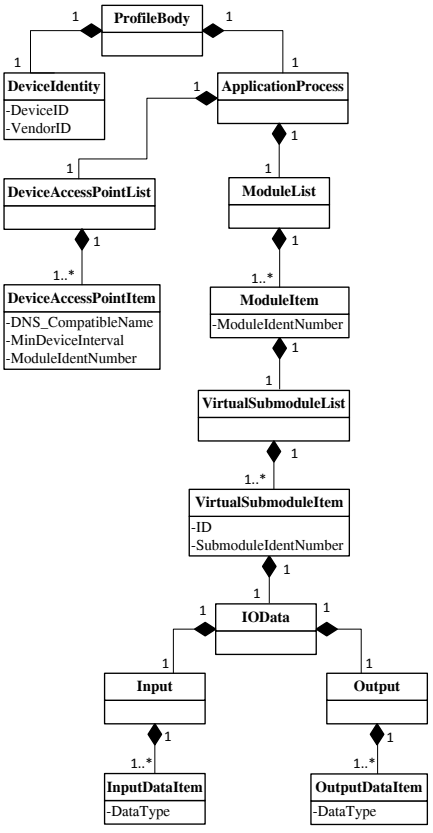

Fig. 2: Structure of a GSD file

An IO-Device is identified by the vendor and device ID stored as an attribute of the *DeviceIdentify* node in the GSD file. A device consists of one Device Access Point (DAP), which is usually the bus connector. Beside the DAP, there are several modules that can offer different functionalities (i.e. digital inputs or outputs). In a block IO-Device, the DAP and the modules are fixed and the configuration of such a block device can be read from the GSD file. In contrast the configuration of a modular IO-Device can be changed. The user can select modules of different types from a pre-defined list and plug them into a physical slot of the IO-Device. In such a case, the GSD file contains a description of all possible DAPs and modules. The actual configuration of the IO-Device must be added to the engineering tool by the user. Each available DAP is described by the *DeviceAccessPointItem* in the GSD file. The modules are part of the *ModuleList* and are defined by the *ModuleItem*. The DAPs and modules are identified by the attribute *ModuleIdentNumber*. The modules can be divided into submodules, described by a node in the *VirtualSubmoduleItem*. *VirtualSubmoduleList* contains all submodules - but in practice, each module has exact one submodule. The definition of inputs and outputs are part of the submodule description. The node *IOData* has the childs *Input* and *Output* and for each of them one or more *DataItem*s are defined. The attribute *DataType* specifies the kind of data and its length (i.e. *Unsigned8*).

The engineering tool is using the provided GSD files and the information about the attached modules of the IO-Devices to configure the IO-Controller that establishes a connection to the IO-Devices afterwards. Section IV describes how this process can be automated.

IV. Autoconfiguration procedure

To achieve PnP, the communication between the IO-Controller and the IO-Devices must start automatically without any manual interventions. Nowadays, the automation engineer must add all devices and their RTE specific DDF in the engineering tool that determines the RTE communication parameters. There are three main challenges to be addressed to automate this procedure:

1) **Discovery**: Every new device must be discovered.
2) **Identification of attributes**: The attributes of the new device must be detected.
3) **Configuration**: The RTE must be configured in accordance to the discovered devices and their attributes.

The proposed architectural component that is responsible for the execution of these steps is called autoconfiguration service. It deals with the first and second challenge by using OPC-UA. The corresponding implementation is described in section IV-A. Details about the third challenge, the configuration of the RTE, is provided in section IV-B.

A. Discovery

The discovery of new devices by OPC-UA is independent from the used RTE system. Therefore, it is necessary that the RTE offers a TCP/IP channel which does not require any special configuration steps. In this case the RTE technology is transparent for standard TCP/IP applications. For instance, this ad hoc channel is available in PNIO, Ethernet/IP and Ethernet Powerlink. The discovery process is part of Fig. 3.

In the first step, the newly connected IO-Device gets an IP address from the DHCP server of the network. After this the OPC-UA server of the IO-Device registers itself at the

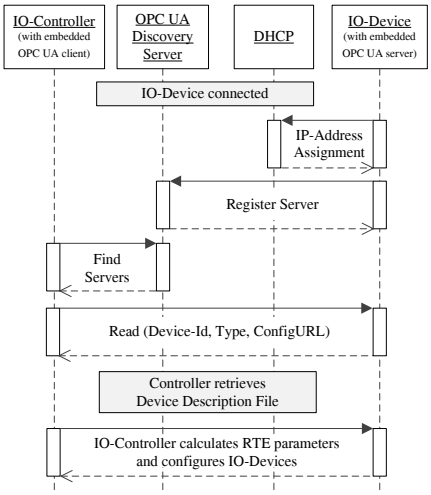

Fig. 3: Autoconfiguration sequence

OPC-UA discovery server, which can be implemented on the same machine as the IO-Controller. Currently, no methods for finding the discovery server automatically by the device are defined by OPC-UA. In our approach the IP address of the discovery server is announced via a site-specific DHCP option [14] to the devices. Using this IP address the OPC-UA server registers the IO-Device at the discovery server. On the other side the OPC-UA client of the IO-Controller gets a list of all registered IO-Devices from the discovery server. The IO-Controller explores the namespace of each IO-Device by a read request and collects its meta data, e. g. the device id. In this new proposal, the DDF is not stored on the device, as in [10], since the autoconfiguration approach of this paper targets very resource limited devices, and modular PNIO devices can have large descriptions. For example, the GSD file of the Siemens ET 200S IO-Device has a size of 2.71 MB. However, there are some alternatives to the local storage of the file. One could be to store an URL to the configuration file in the ConfigURL variable of the OPC-UA namespace of the IO-Device. Another possibility is a central storage place in the network or on the IO-Controller. Then, the appropriate file can be identified by the device id, which is a part of the OPC-UA meta data. The next steps are RTE dependent and are described in section IV-B.

B. Configuring the RTE

While discovery is independent from the used RTE, this is obviously not true for the determination of the RTE communication parameters. In this work PNIO is used as an example. After the discovery phase the GSD files of all IO-Devices are known to the autoconfiguration service. However, it is not aware of the modules attached to the IO-Devices. Its task is now to collect these information, to parse all GSD files, to determine the PNIO parameters and to configure the IO-Controller.

1) Identification of attached modules: In [15], the installed modules of an IO-Device are identified by the Simple Network Management Protocol (SNMP). This method is comparatively complex and SNMP is not mandatory for all PNIO devices. In our approach the PNIO command *Read Implicit Request* with the option *RealIdentificationData for one API* is used. This command must be supported by all PNIO devices. Furthermore, the request consists of only one frame, and a preceding connection establishment is not necessary. The addressed IO-Device answers with its identification data. The response is visualized in Fig. 4.

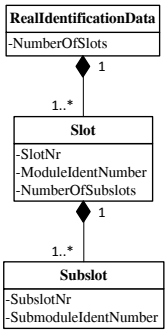

Fig. 4: Structure of the *RealIdentificationData* response

For each slot of the IO-Device with a plugged module, the *RealIdentificationData* structure contains one *Slot* node. The attributes of this node contain the slot number and the identification number of the connected module. The latter is a link to the correspondent *ModuleItem* in the GSD file. With this information the autoconfiguration service is aware of all connected modules and their descriptions in the GSD file. Furthermore, a module might consist of several submodules. The submodules are linked to the subslots in the same way by the node *Subslot*.

2) Determination of PNIO parameters: As a first step device names and IP addresses of the IO-Devices must be set. The autoconfiguration service uses the attribute *DNS_CompatibleName* from the GSD file as device name. As IP address a randomly chosen address from the IO-Controller's subnet is assigned after the IO-Controller has checked the availability via the address resolution protocol. Both device name and IP address are set via the Discovery and Basic Configuration Protocol (DCP) [10]. Furthermore the autoconfiguration service must provide the IO-Controller with the needed data for the connection establishment which is initiated by a connection request frame. The structure of this frame is shown in Fig. 5.

Between the IO-Controller and the IO-Device two IO communication relations (IOCR) are established. The input IOCR describes the transmission of process data from the IO-Device to the IO-Controller. The input from all modules which have to send data to the IO-Controller is merged into one frame. The input IOCR defines the send cycle of this frame and the order of the data within the frame. The attribute *SendClockFactor* is derived from the *MinDeviceInterval* in the GSD file. Details on the computation can be found in [16]. For each *InputDataItem* from the GSD file one *IODataObject* as a child of the input IOCR block is created. The attributes *SlotNr* and *SubslotNr* tell the IO-Device which data is addressed by this *IODataObject*. The slot and subslot number are known to the autoconfiguration service due to the IO-Device's *RealIdentificationData* response. In addition the offset of the

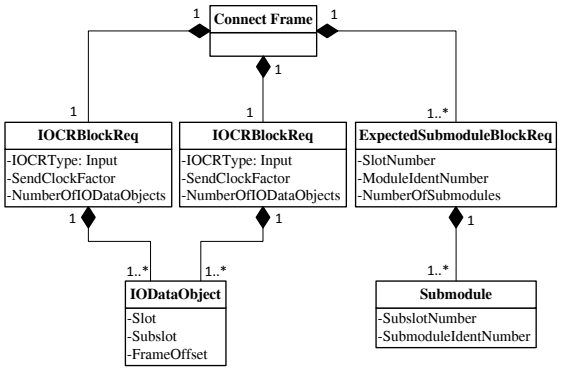

Fig. 5: Simplified structure of the connection request frame

IO data in the input IOCR frame is defined. Therefore, the autoconfiguration service must be aware of the length of the IO data. This information is available in the *DataType* attribute of the *InputDataItem* node in the GSD file. The approach of defining the input IOCR block can be transferred accordingly to the output IOCR block for the IO data from the IO-Controller to the IO-Device.

The *ExpectedSubmoduleBlockReq* is intended to ensure that the actual module configuration of the IO-Device matches the settings in the engineering tool made by the user. Therefore, this block contains information about which modules are expected at the specific slots of the IO-Device. If the answer of the IO-Device reports a module misconfiguration, the autoconfiguration process was faulty and an user intervention will be necessary.

In Fig. 6 an example of the parameter determination is shown. The arrows indicate how the autoconfiguration service combines the data from the GSD file and the *RealIdentificationData* response. In this example, the IO-Device consists of two modules with one submodule each. For the sake of clarity only the most important arrows are shown.

V. Validation

The functionality of our approach has been tested using the setup shown in Fig. 7.

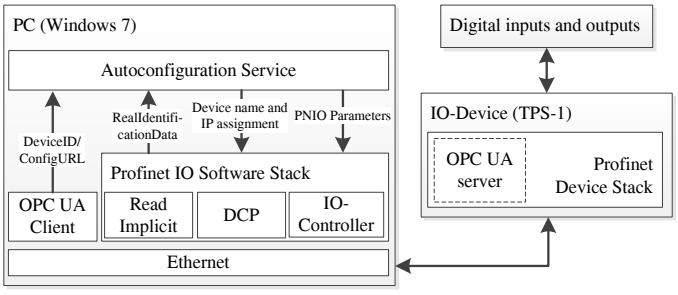

Fig. 7: Setup for autoconfiguration validation

The autoconfiguration service is implemented in C#. The OPC-UA client is based on the reference architecture from the OPC foundation [17]. For all PNIO functions the PNIO software stack from KW Software [18] is used. The IO-Device is built onto the top of the PNIO single-chip solution TPS-1 [19]. The TPS-1 is connected to digital inputs and outputs to show the proper process data exchange.

The OPC-UA server functionality is implemented as an extension of the firmware of the TPS-1 by introducing a tailored OPC-UA stack to transport the messages in binary encoding on top of a micro TCP/IP stack. For the server functionality the ”Nano Embedded Device Server Profile”, as specified by the OPC Foundation for chip level devices, is used. After a thorough analysis of the OPC Foundation software stack, mandatory design components of this profile have been identified and a corresponding software stack was implemented completely from the scratch for this work. This new implementation provides a reduced complexity and down-scaled footprint. For the OPC-UA server function together with a micro TCP/IP stack, only 15 KB of memory footprint is required to implement a basic application. The protocol stack is implemented in ANSI C and consists of around 2000 lines of code.

When the modified TPS-1 is connected to the network, the URL of its GSD file is transferred via OPC-UA to the autoconfiguration service. Afterwards the service invokes the *Read Implicit* command from the PNIO software stack and parses the response. The PNIO parameters are determined and stored in an XML configuration file for the IO-Controller of the stack. The controller contains a counter as an example application to drive the digital outputs of the IO-Device. On the other side the state of the digital inputs is visualized on the PC by the IO-Controller.

VI. Conclusion and Future Work

The autoconfiguration of real-time Ethernets is an important step towards more flexible systems in the domain of industrial automation. This work takes existing approaches as a basis and extends them. The autoconfiguration process can be divided into the parts discovery and RTE configuration. Here OPC-UA is proposed as the technology for discovering of new devices and for exchanging basic device information. These functions are realized using standard TCP/IP over a not configured ad hoc channel. In contrast to other solutions, OPC-UA is suitable for small resource constrained devices. The information gathered by OPC-UA are further processed by an RTE specific autoconfiguration service. In this work Profinet IO is used to implement a prototypical example. Also modular Profinet IO devices are supported here. Their module configuration is ascertained by a special PNIO command. The answer to this command is merged with the GSD file of the correspondent device to determine all necessary parameters for the Profinet IO communication.

While this work extends the use cases for autoconfiguration of Profinet IO, the isochronous variant Profinet IRT is still not supported, because the communication planning in such a system is much more complex. There is a first approach to simplify the scheduling of Profinet IRT [20] which will be integrated into a autoconfiguration solution as a part of future work.

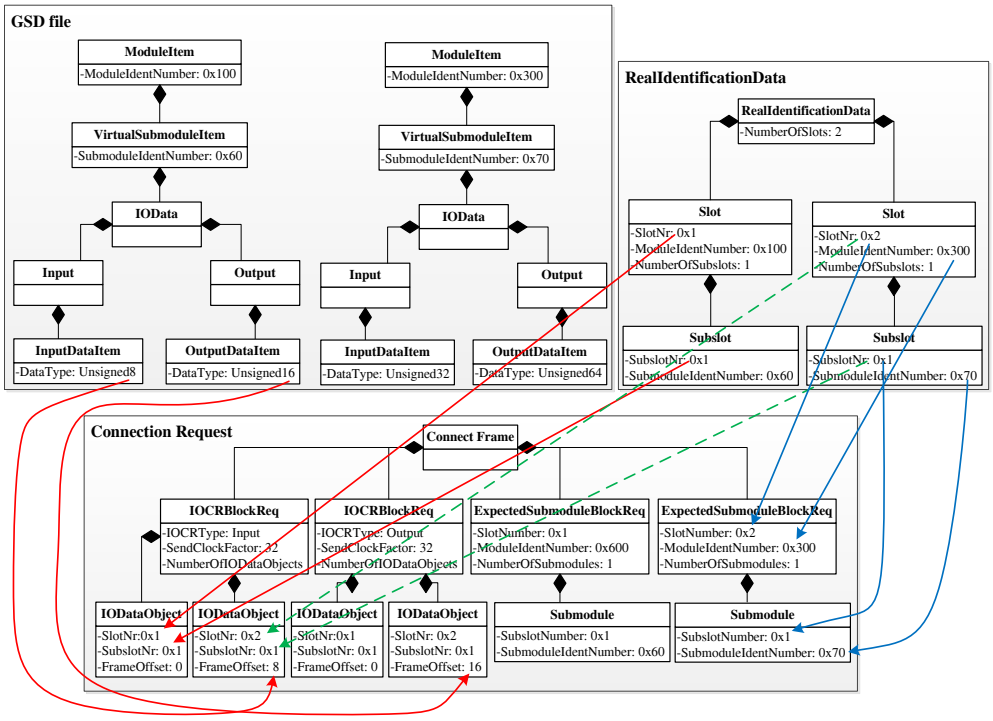

Fig. 6: Example for determination of Profinet IO parameters

ACKNOWLEDGMENT

This work was partly funded by the EU FP7 STREP project IoT@Work under grand number ICT-257367 as well as by the German Federal Ministry of Education and Research (BMBF) within the Leading-Edge Cluster "Intelligent Technical Systems OstWestfalenLippe" (it's OWL).

REFERENCES

[1] A. W. Colombo and R. Harrison, "Modular and collaborative automation: achieving manufacturing flexibility and reconfigurability," *International Journal of Manufacturing Technology and Management*, vol. 14, pp. 249–265, Mar. 2008.

[2] Universal Serial Bus Specification. [Online]. Available: http://www.usb.org/developers/docs/

[3] UPnP Forum homepage. [Online]. Available: http://www.upnp.org/

[4] Devices Profile for Web Services Version 1.1. [Online]. Available: http://docs.oasis-open.org/ws-dd/dpws/1.1/os/wsdd-dpws-1.1-spec-os.pdf

[5] D. Mora, M. Taisch, A. Colombo, and J. Mendes, "Service-Oriented Architecture approach for industrial system of systems: State-of-the-art for energy management," in *10th IEEE International Conference on Industrial Informatics (INDIN'12)*, Beijing, China, Jul. 2012.

[6] A. Cannata, S. Karnouskos, and M. Taisch, "Evaluating the potential of a service oriented infrastructure for the factory of the future," in *8th IEEE International Conference on Industrial Informatics (INDIN'10)*, Osaka, Japan, Jul. 2010.

[7] S. Hodek and J. Schlick, "Ad hoc field device integration using device profiles, concepts for automated configuration and web service technologies: Plug&Play field device integration concepts for industrial production processes," in *9th International Multi-Conference on Systems, Signals and Devices (SSD'12)*, Chemnitz, Germany, Mar. 2012.

[8] J. Imtiaz, J. Jasperneite, K. Weber, F.-J. Goetz, and G. Lessmann, "A novel method for auto configuration of realtime Ethernet networks," in *13th IEEE International Conference on Emerging Technologies & Factory Automation (ETFA'08)*, Hamburg, Germany, Sep. 2008.

[9] G. Reinhart, S. Krug, S. Hüttner, Z. Mari, F. Riedelbauch, and M. Schlögel, "Automatic configuration (Plug & Produce) of Industrial Ethernet networks," in *9th IEEE/IAS International Conference on Industry Applications (INDUSCON'10)*, São Paulo, Brazil, Nov. 2010.

[10] L. Dürkop, H. Trsek, J. Jasperneite, and L. Wisniewski, "Towards autoconfiguration of industrial automation systems: A case study using Profinet IO," in *17th IEEE International Conference on Emerging Technologies & Factory Automation (ETFA'12)*, Kraków, Poland, Sep. 2012.

[11] L. Dürkop, J. Imtiaz, H. Trsek, and J. Jasperneite, "Service-oriented architecture for the autoconfiguration of real-time Ethernet systems," in *3rd Annual Colloquium Communication in Automation (KommA'12)*, Lemgo, Germany, Nov. 2012.

[12] *IEEE standard for Ethernet - section 2*, IEEE Std. 802.3, 2012.

[13] *IEEE standard for local and metropolitan area networks - media access controll (MAC) bridges and virtual bridged local area networks*, IEEE Std. 802.1Q, 2011.

[14] S. Alexander and R. Droms, "DHCP options and BOOTP vendor extensions," RFC 2132, Mar. 1997. [Online]. Available: http://www.ietf.org/rfc/rfc2132.txt

[15] D. von Rohr, M. Felser, and M. Rentschler, "Simplifying the engineering of modular Profinet IO devices," in *16th IEEE International Conference on Emerging Technologies & Factory Automation (ETFA'11)*, Toulouse, France, Sep. 2011.

[16] M. Popp, *Industrial communication with Profinet*. Karlsruhe: Profibus Nutzerorganisation, 2007.

[17] OPC foundation homepage. [Online]. Available: http://www.opcfoundation.org/

[18] KW Software Profinet Communication Stacks. [Online]. Available: https://www.kw-software.com/en/profinet-industrial-ethernet/communication-stacks

[19] Profinet IO Device Chip TPS-1. [Online]. Available: http://www.renesas.com/products/soc/assp/fa_lsi/ethernet/tps1/index.jsp

[20] L. Wisniewski, M. Schumacher, J. Jasperneite, and S. Schriegel, "Fast and simple scheduling algorithm for Profinet IRT networks," in *9th IEEE International Workshop on Factory Communication Systems (WFCS'12)*, Lemgo, Germany, May 2012.