COMMUNICATION, CONFIGURATION, APPLICATION:

The three layer concept for Plug-and-Produce

Uwe E. Zimmermann, Rainer Bischoff
KUKA Roboter Gmbh, Zugspitzstraße 140
86165 Augsburg, Germany
UweZimmermann@kuka-roboter.de, RainerBischoff@kuka-roboter.de

Gerhard Grunwald, Georg Plank, Detlef Reintsema
German Aerospace Center (DLR), Institute of Robotics and Mechatronics, Oberpfaffenhofen, 88234 Wessling Germany
Gerhard.Grunwald@dlr.de, Georg.Plank@dlr.de, Detlef.Reintsema@dlr.de

Keywords: Plug&Play, robotics, reduced set-up time, determinism, real-time communication, device description

Abstract: The Plug-And-Play is a synonym for the simple use of computer systems in office applications. This appealing idea also emerges in the branches of robotics and automation. But due to their specific technical and application driven requirements i.e. real-time, determinism, safety… a simple taking over of the technology is not sufficient. Plug-and-Produce extends the "easy to use" for applications in robotics and automation. This will lead to a dramatically reduction of set-up times of work-cells. In order to fulfil the various requirements a three layered Plug-and-Produce architecture is presented. The communication layer addresses all topics regarding the bus-system level; the configuration layer is responsible for the control and operation system level; and the application layer addresses the needs of the user and system programmer. A prototypical implementation is also presented.

1. INTRODUCTION

Technically, the integration of components i.e. a memory stick, a camera, or a disk drive into a computer system is a very complex task. Different kind of knowledge is required. This includes hardware, communication systems, operating systems, programming and application interfaces. Practically, the user wishes the simple addition of his new device without requiring reconfiguration or manual installation of device drivers. "Plug-And-Play"(PnP) is the synonym for this wish. In the early 1980s the NuBus (NuBus, 1983) which was originally developed by MIT and used by Apple Macintosh and Texas Instruments was one of the earliest PnP-busses. Today the Universal Serial Bus (USB) is the widest spread and probably best known PnP-bus (Axelson, 2005).

The Plug-And-Play technology was developed for the simple use of computer systems in office applications. Recently this appealing idea also emerged in the branches of robotics and automation. But due to their specific technical and application driven requirements (details see in section 2) a simple taking over of PnP is not possible. Hence the EU funded project *SMErobot*™ (SMErobot, 2005) created the term „Plug and Produce"(PnProduce) to express these peculiarities and to delimit from the office applications.

The potential of PnProduce is very obviously: The set-up time of robot cells is time-consuming and thus an important cost factor. It take days or even weeks just to get all peripheral systems communicating and working with each other, even though the system was planed and designed very detailed in advance. Costs could be dramatically reduced, if it is possible to reduce the installation time. Especially future robotics systems will require much more flexibility due to frequent task and equipment changes. It should be possible to connect a new, even a priori unknown device to the robot controller and to use it immediately without the need of a long and error prone manual configuration.

Besides the reduction of costs a further important factor is the "easy to use" feature. In future, more and more robots will find their way into small and medium enterprises, into the public and the health

sector. Also service robots or assistants for entertainment and at home are emerging. These systems will be in common, that non-robotic experts will have to deal with such complex systems. Adding of new tools, exchanging of faulty devices or upgrading the system has to be manageable without the need of an expert.

By means of scenarios and use cases section 2 motivates the details of PnProduce and specifies the inherent technical requirements. In section 3 we introduce the three layer concept of PnProduce and. Some first implementation results are presented in section 4.

2. PLUG AND PRODUCE REQUIREMENTS

PnProduce concepts for industrial applications especially robotics can adapt existing PnP concepts from IT. But there are some special issues that have to be covered and which are not solved by available solutions. By means of use cases some features of PnProduce are discussed and also the specific technical requirements.

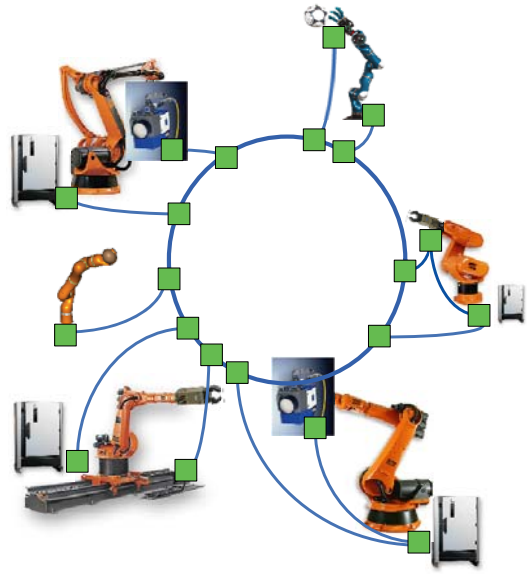

Figure 1: Scenarios for PnProduce

2.1 Use Cases

Figure 1 shows a generic setup of a work cell with multiple, different robots, tools, grippers, hands, sensors, and controllers. They are connected via real-time bus systems i.e. EtherCAT, Sercos …. Also the single components inside a robot should be connected via PnProduce.

2.1.1 Device exchange

A robot component i.e. a drive unit has to be exchanged or upgraded. The servicing staff powers off the robot, exchanges the devices. In case of PnProduce the robot has only to be switched on being operational again. The central control unit automatically identifies the new device, reads the actual device status information, configures it and integrates its functionality into the application.

2.1.2 (Automatic) tool change

Especially for versatile robot assistants frequent tool changes are a common requirement. A device driven approach is just a subversion of the previous use case, where a new tool (e.g. driller) is plugged to the robot and the associated tool service (e.g. “drill”) is available for the user.

In a service driven approach the user first selects the service (e.g. “drill”) and then the tool that could perform the task is automatically selected by the robot system itself. This means that an automatic tool changer system has to be set up.

2.1.3 Start up of a new system/application

In general an application can be characterized by the potential functionality of the connected components and devices. When powering on the functionalities of the devices are communicated with the central controller or a programmer. Bus, devices and robot controller are already configured for the immediate use. The application itself can be described by the detailed device configurations needed for running the task. This could be done manually or (semi-)automatic.

2.1.4 Start up of an existing system/application

An application as described above may be stored in a file. When switching on the system the controller gets all status information from the devices which are connected via the real-time bus. These data can be compared with the stored description of the planned application. The controller can verify whether all devices are connected and properly configured. If not appropriate feedback to the user can be given.

2.2 Deterministic and Real-time Communication

The communication in robotics and automation must guarantee some features to facilitate a proper and safe operation of the system. One of the most important features are real-time and determinism. As there are multiple specifications on the term real-time, we will use in the context of PnProduce: “The timing constraints of the system must be guaranteed to be met. Guaranteeing timing behaviour requires that the communication is predictable”. The term deterministic tightens the requirement in real-time communication. It is not sufficient to guarantee the data within a time frame. The data have to be delivered at a precise time.

2.3 Static and dynamic parameters

In the past, the application programmer differentiates cyclic and acyclic communication. Cyclic data are i.e. the sensor values from a force torque sensor or the nominal and real value of the robot joints. These data types are also called “process images” (CAN, 2007), that are a complete or partial local data base. The application or the devices operate only on this local data, whereas the bus system is responsible to keep the distributed data consistent. The advantage is that communication and application are highly decoupled. Another important factor is the deterministic exchange of data as the bus load is constant and could be analyzed a priori.

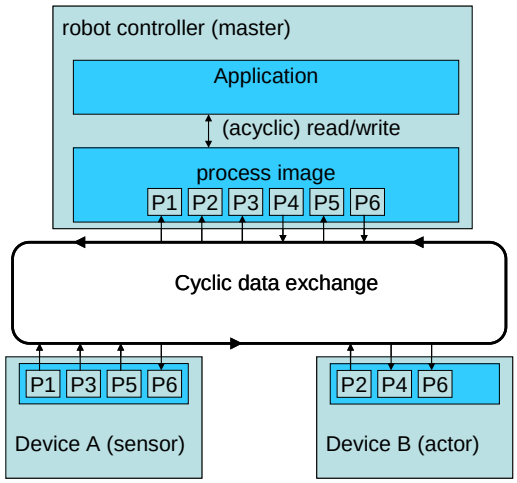

Figure 2: Cyclic/acyclic data exchange

Also there is an acyclic communication regarding the application. Data types are commands or events. Commands are triggered by the application program and are explicitly programmed by the user. Examples are “gripper open” or “sensor on”. They are typically for actuators and tools that are connected to the robot controller. The philosophy of commands is very similar to the Service Oriented Architecture (SOA) approaches. Most existing robot languages are command based and therefore PnProduce concepts could be integrated seamlessly.

Events are used for the asynchronous transfer of small data sets to inform the control system or the devices on their actual status. They may also be used for setting parameters of the attached devices and/or software.

The parameters may be classified in static and dynamic parameters. The latter may be set during system start-up or in case of configuration purposes. But they cannot or should not be changed during run-time. Thus there is no need to include these parameters in the process data sets. In general, static parameters are device specific. Dynamic parameters dynamically lead to more flexible systems. For example, if a gripper with an adjustable gripping force is used, the force parameter can be set once (configuration data) before the application starts or can be changed depending on what kind of object has to be gripped (process data).

Commands and events are still called acyclic even it is communicated in a cyclic manner on the bus level. The process images as well as the data exchange between the local data bases has to be configured, which is a complex task and leads to high demands for a PnProduce system. This is especially true if the data exchange should be dynamically changed.

2.4 Plug-And-Play Variants

There are different grades in the performance of Plug-And-Play systems. The simplest version is *Cold PnP:* All devices and components are switched off and get connected. Thereafter the entire system is switched on.

The most difficult variant is *Hot PnP*. The entire system is in operational mode. The user may remove or attach a device without further intervention and most important without disturbing the running application.

An intermediate level is *Coordinated PnP* which is a preliminary stage of Hot PnP. The chance nature of attaching/removing devices is eliminated. This guarantees that the application itself is not disturbed. The attaching/removal is user or program controlled.

Another classification scheme for PnP is complete, semi-automatic, or configurable.

Complete PnP:

- The attached device is automatically recognized and may be used without further intervention.
- Device descriptions are stored in a data base; drivers are automatically configured and integrated to the application system.

Semiautomatic PnP:

- The device is attached and device type/class is recognized.
- User has to integrate the appropriate drivers (CD, DVD, Internet …)
- No further configuration is needed

Configurable PnP:

- Additionally to semiautomatic the user has to configure the drivers, devices manually.

2.5 Abstraction and Descriptions

Hot-PnP, Complete-PnP, and the other variants presuppose the description of devices, configurations, and applications. The purpose is to encapsulate some device/configuration/application dependent functionality thus it can be used in a modular fashion.

Device Descriptions are a key technology for PnProduce. There are a several approaches which will be shortly presented.

The first is EDDL, the Electronic Data Definition Language and its predecessor the GSD, the Generic Station Description. Both of them are used in the PROFIBus Setting (EDDL, 2007).

The GSD is the obligatory “ID card” for every Profibus device. It contains the key data of the device, details relating to its communication capabilities and other information relating, for example, to diagnosis values. In the device integration process the GSD is sufficient to employ for the cyclic exchange of measurement values and manipulated variables between the field device and the automation system.

The EDDL is a textual and OS independent format to describe field devices. It gives hints on how to model the GUI, which is used to configure the devices. This results in a standardized GUI for devices of every manufacturer. The EDDL holds fields for metadata, such as ordering and maintenance information. The standardization is IEC 61804-2.

Another example for a description language is EDS and DCF. EDS stands for electronic data sheet and is used with the CAN bus, mainly with CANopen and DeviceNet. EDS is just a template, DCF, device configuration file, is a concrete instance of this template.

EDS is also a plain ASCII or XML file, standardized in ISO 15745. By means of an EDS, a device can be described with respect to the content of its object dictionary. User defined data types (records), their value(s), simple values of predefined type, arrays, "funktion pointers” and similar data is stored there and therefore described in the EDS. There are a lot of EDS Files predefined for devices supporting the CAN bus (CAN, 2007).

FDCML is the field device configuration markup language. It describes identity, logical and physical communication facilities (bus independence), functionality, and configuration facilities. Also it supports the multilingual documentation of the device. It's flexible in respect to future developments and is capable of describing dependencies between device parameters. It is possible to describe the different types of devices which can be connected to this device, the needed resources to use this device, its structure and so on (FDCML, 2007).

The descriptions used in UPnP are written in XML syntax (therefore again multilingualism possible). UPnP describes identity, provided services (functional units within devices), actions (functional units within services) and state variables related to these services. For each service the description contains the type and URI's for eventing and controlling. The device itself contains a URI for its presentation. With these URIs it is possible to interact with the service and to examine the device (UPnP, 2007).

The actions are parameterized with supplied arguments, which are defined within the service. These arguments can be in or out arguments, relate to any state variable within the service and have a defined return value.

The state variables are typed, have a value range and can send events. It is a very abstract approach, so it doesn't include any description of physical connections and therefore bus-independent.

Used in our setting the abstract approach and the missing definition of the physical connection display the need for extensions of the description.

Configuration descriptions are another category of *descriptions*. They describe how a unique instance of a device is actually used. Configuration descriptions are only valid for exactly one device instance and often are dependant from a single application. A configuration description is more or less just a “snapshot” of the actual device state, e.g. the values of the static parameters. They can be used

to make device data persistent, to check if the actual configuration is valid and for automatic configuration of devices after a device exchange.

The application description contains the devices and services needed to run an application. It is used to check, if all conditions are met to start the application or if there are modifications regarding the connected devices.

3. THREE LAYER CONCEPT

It is obviously to see that PnProduce has different levels of abstraction. You can identify three "Plug-And-Produce"-layers to meet the requirements of robotics and automation systems:

1. Communication and bus-system level: log-in and log-out of subscribers;
2. Control and operation system level: integration and release of components in the environment;
3. Application level: the offered functionalities of the subscribers are used.

The German national project PAPAS: Plug And Play for Automation Systems (PAPAS, 2006) addressed this complexity and suggested a layered structure similar to the famous ISI/OSI layers for communication. Figure 3 relates the PAPAS structure both to the ISO/OSI model and the typical field bus approach.

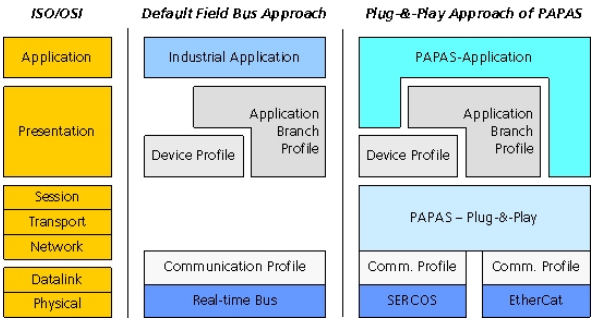

Figure 3: The PAPAS Plug-And-Play layer architecture (Plank et.al. 2006)

The Datalink and Physical Layer are directly related to the communication bus and its physics. Examples of a field-bus are CAN or Profibus. PAPAS investigated among others Sercos (SERCOS, 2007) and the Ethernet-based real-time busses EtherCAT (ETG, 2007) and Ethernet-Powerlink (EPSG, 2007). The two layers are the only, clear delimited layers regarding the ISO/OSI specification. The layers Session, Transport, and Network are not used in the general field-bus approaches. In all three models the application and presentation layers are oriented towards the application and/or the configuration of the application. The limits of these ISO/OSI layers are indistinct.

3.1 Communication PnProduce

The basis of PnProduce is the communication layer, as it addresses the potential (i.e. bandwidth, load, determinism, clock …) of the communication itself. It is directly related to Layer 1 to Layer 5 of the ISO/OSI reference model. The PAPAS Plug&Play layer addresses here the PnP-functionality regarding the bus-systems and their specific communication protocols. At this level the independency of the bus-system is implemented.

When these layers are successfully switched on the systems knows all the participants by its network address but not by its function. The transferring data types are not known yet which guarantees the independence of the application.

3.2 Configuration PnProduce

The effect of this layer is twofold. It configures the system with regard to the communication as well as to the application.

When the communication system is operational, the participants are known and the master is able to read their device descriptions (see 2.5). Typical device information is:

- Synchronous, asynchronous communication
- Minimal/maximum bandwidth for sending and/or receiving data i.e. joint value, force/torque value, image data …
- Minimal/maximum clock for sending and/or receiving data

According to their requirements and the needs of the specific application the communication system can be configured.

Analogue the devices and controls have to be configured regarding their functionality. Following tasks can/should be done in this layer:

- Determining which devices i.e. sensor are available;
- Determining which functions, services, and parameters are available (see device descriptions);
- Automatic reconfiguration of a device after it has been exchanged because the old one was defect;

- Reading device status information i.e. error log, software release, operating hours, and other statistical data.

3.3 Application PnProduce

The intention of this layer is to structure the open ended world of applications thus you can reuse robot programs or tasks (synonym for robot applications) in another context or environment. A robotic task could be i.e. to “drill a hole”. This task is composed by a robot, a sensor, a drilling machine, a piece of wood, and software commanding the active devices. You can repeat the task with the same components but also with i.e. another robot. The application remains the same. Only the functionality of the connected devices is of interest.

The following scenarios describe a graded structure, which can be distinguished within Application PnProduce.

- Standardised Functions/Services: If functions, services and parameters are standardised like CANopen profiles (Pfeiffer et.al. 2003, CAN, 2007), they can be used automatically and called from the system or the application.
- User integration: If a new device with an unknown functionality is attached, the system is able to autonomously detect and integrate it at the communication level. But it is not able to integrate its services or functions. It will inform the user on the new component its offered functionalities.
- Semantic information: Another possibility is to integrate semantic knowledge into unknown functionality. But this is a very difficult and not completely solved task. Besides the syntactic description a service the meaning must also be represented.

Whereas the application PnProduce layer is fully independent from the communication PnProduce layer and vice versa, the configuration PnProduce layer can have dependencies with both, communication and application.

3.4 PnProduce Layer Model

The following will explain the PnProduce layers and their interdependencies in more detail.

The new high performance communication systems must meet the modern robotics and automation demands:

- Distributed automation
- Close and coordinated coupling of devices with different functions, timing, and data loads
- High clock speed
- Short reaction time
- Reduction of costs

The actual serial real-time bus systems (i.e. EtherCAT, Ethernet-Powerlink, and Sercos) comply with these demands. But they differ in transfer modes and protocols. Thus the integration of different devices equipped with different bus systems into one robotics application is very difficult and needs a lot of expert knowledge. In order to reach the Communication PnProduce functionality and thus a transparent and deterministic access to the bus system level, an abstraction of the PnP-behaviour from the bus-system is needed.

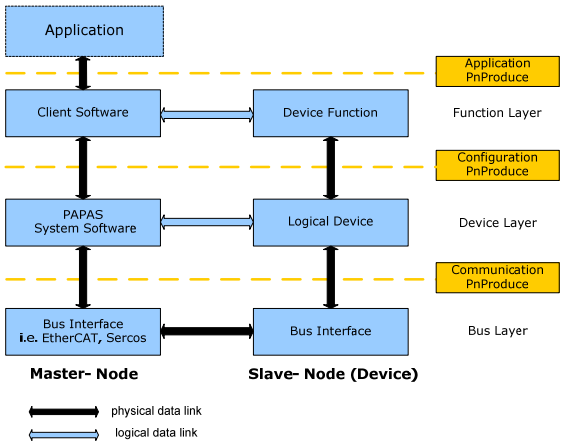

Figure 4: The PnProduce layer model. Each transition interface provides a specific kind of PnProduce-capability.

PnProduce assumes a master/slave architecture which is commonly used in robotic and automation systems. The PnProduce layer model (Figure 4) shows the most important components and the data flow.

- Master-Node: i.e. robot controller;
- Slave-Node: i.e. sensor device in its entirety;
- Device Function: description of the input/output characteristics i.e. sensor values;
- Client Software: API of the device;
- Logical Device: Basic interface, which facilitates the master an unified handling of the varying devices;
- PAPAS System Software: Specific w.r.t. the operation system but independent of the client software and the bus interface;
- Bus-Interface: hardware i.e. EtherCAT, Sercos

Master and slave are composed of three layers. The Bus Layer provides the physical and packet oriented connection. The Device Layer contains all the

system information needed for the device configuration. The Function Layer describes the logical interconnection between master and slave.

The master puts a *request* on the bus. If the needed communication resources are available either the master or the slave sends its data. The success is indicated by *acknowledge*. The PnProduce protocol provides four data transfer modes:

1. *Control*: configuration of new attached devices and device control. All devices have to support this mode.
2. *Interrupt*: Asynchronous data transfer of small data sets. The protocol guarantees the data transfer in device specific service intervals.
3. *Bulk*: Asynchronous data transfer of large data sets. The protocol guarantees the data transfer as soon as the needed bandwidth is available.
4. *Isochronous*: This mode guarantees fixed data rates and a maximum latency. When a new device is attached the master verifies whether the requested communication bandwidth is available.

Figure 5: A light-weight robot arm as an assistant for human-like working in industrial environments the PAPAS experimental setup is designed to demonstrate a typical example of an industrial manufacturing process.

4. EXPERIMENTAL SYSTEMS

This section describes in short the first version of an implemented PnProduce system. Here the focus is mainly on the communication PnProduce functionality. The application addressed within the PAPAS experimental setup is based on a robot assistant for human working in industrial environments (see Figure 5). One chosen task of the robot is a typical example of an industrial manufacturing process, which is still executed almost completely manually today, with a very low degree of automation. The assembly algorithms developed and tested on a set of planar objects with complex, non-convex geometric forms are described within (Stemmer et.al, 2006).

Figure 6: Typical classes of slave tools and devices for robotics and automation applications: a gripper with parallel jaw motion, a force torque sensor and a programmable focusing optics (PFO) which have been prepared to interact as an EtherCAT slave device.

For verification of the PnProduce approach a set of typical industrial tools has been prepared to demonstrate hot plugging. Typical tools are a Programmable Focusing Optics (PFO), a compliant force-torque sensor or a parallel gripper (see Figure 6). The PFO of Trumpf Laser flexible welds spots and seams without moving the workpiece, neither the focusing optics having to be moved at all. The PFO focuses the laser beam to any defined location in the working area.

Especially for assembly and part mating with industrial robots, a compliant force-torque sensor was developed. The compliant sensor does not only yield the forces and torques but also the positional/rotational displacements of a tool, e.g., under the influence of gravity.

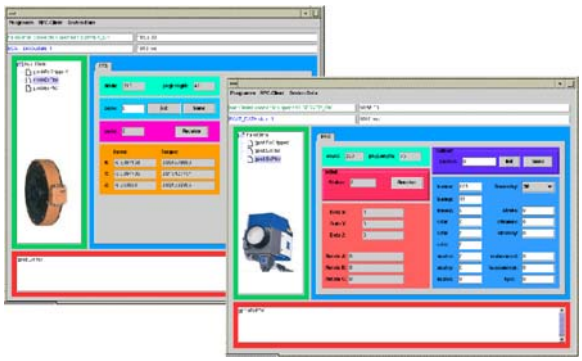

Figure 7: JAVA-based tools can be used to monitor the plug-and-play process, to display recorded device data and to command plugged devices using a general purpose class-specific interface.

With the *Ethernet Control Automation Technology* (EtherCAT) a new Ethernet-based field bus system was chosen at the drive or I/O level (ETG, 2007) for real-time transportation of process data.

The experimental set up uses a line structure configuration. A chain of EtherCAT slaves can be connected in any sequence to the EtherCAT master which monitors the Ethernet bus for plugged devices. Inserting a new device into an expansion slot forces the master to trigger the PnProduce process of the overlaying PnProduce layer without requiring reconfiguration or manual installation of device drivers.

5. SUMMARY AND OUTLOOK

The integration of industrial components i.e. gripper, force/torque sensors, intelligent tools… into a robot work-cell is a very complex task. Different kind of knowledge is required. This includes hardware, communication systems, operating systems, programming and application interfaces. Plug-and-Produce extends the very well known concept of Plug-And-Play towards the use in industrial environments. Due to their specific technical and application driven requirements i.e. real-time, determinism, safety… a simple taking over of the PnP technology is not sufficient. Plug-and-Produce extends the "easy to use" for applications in robotics and automation. This will lead among other things to a dramatically reduction of set-up times of work-cells.

In order to fulfil the various requirements a layered Plug-and-Produce architecture was presented. The key feature is to properly encapsulate the functionalities of one layer with regard to the neighbouring. Standardized interfaces i.e. the device description will permit the interchangeability of hardware and software

Three "Plug-And-Produce"-layers were specified to meet the requirements of robotics and automation systems:

1. Communication and bus-system level: log-in and log-out of subscribers;
2. Control and operation system level: integration and release of components in the environment;
3. Application level: the offered functionalities of the subscribers are used.

The first implementation results especially the Communication PnProduce are very promising. The future activities of Application PnProduce may be influenced by the research in Service Oriented Architectures (SoA). Here we have to investigate whether these concepts satisfy the PnProduce requirements and how they can be integrated.

ACKNOWLEDGEMENTS

This work has been partially funded by the German Collaborative project PAPAS under grant no. 02PH2060 and the European Commission's Sixth Framework Programme under grant no. 011838 as part of the Integrated Project SMErobot™.

REFERENCES

NuBus, 1983. *NuBus Specification,* Texas Instruments.

Axelson, J. 2005. *USB Complete: Everything You Need to Develop Custom USB Peripherals.* Lakeview Research, Madison WI.

SMErobot, 2005. *SMErobot™* - The European Robot Initiative for Strengthening the Competitiveness of SMEs in Manufacturing, www.smerobot.org, Integrated project funded under the European Union's Sixth Framework Programme (FP6).

CAN, 2007. *Controller Aera Network*, www.can-cia.de.

EDDL, 2007. *The Electronic Device Description Language (EDDL)*, www.eddl.org.

FDCML, 2007. *Field Device Configuration Markup Language*, www.fdcml.org.

UPnP, 2007. *Universal Plug and Play*, The UPnP™ Implementers Corporation, www.upnp-ic.org.

PAPAS, 2006. *Plug-And-Play Antriebs- und Steuerungskonzepte für die Produktion von morgen* (in German), www.robotic.de/PAPAS.

Plank, G., Reintsema, D., Grunwald, G., Otter, M., Kurze, M., Löhning, M., Reiner, M., Zimmermann, U., Schreiber, G., Weiss, M., Bischoff, R., Fellhauer, B., Notheis, T., Barklage, T. 2006. PAPAS Abschlussbericht (in german), http://www.dlr.de/rm-neu/Portaldata/52/Resources/dokumente/papas/PAPAS-Abschlussbericht.pdf

ETG, 2007. *EtherCAT Technology Group: „Technology Introduction and Overview"*, http://www.ethercat.org.

SERCOS, 2007. *SERCOS interface*, www.sercos.de.

EPSG, 2007. *Ethernet POWERLINK Standardization Group*, www.ethernet-powerlink.org.

Pfeiffer, O., Ayre, A., Keydel, C. 2003. *Embedded Networking with CAN and CANopen.* RTC Books

Stemmer, A., Schreiber, G., Arbter, K., and Albu-Schäffer, A. 2006. *„Robust Assembly of Complex Shaped Planar Parts Using Vision and Force"*, in IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI2006), 3.-6. September 2006, Heidelberg, Germany.