Modular and Generic Control Software System for Scalable Automation

Christian Brecher, Martin Freundt, Daniel Schöllhorn

Fraunhofer Institute for Production Technology, Steinbachstrasse 17,
52074 Aachen, Germany, Tel.: +46(0)241-8904-253, Fax.: +49(0)241-8904-6253,
E-mail: martin.freundt@ipt.fraunhofer.de

Abstract. The development of automated production systems is subdivided in two mayor tasks. One is the development of the processes needed to meet the requirements for the product, the other is the setup of a control system enabling the hardware to perform these processes. Typically the larger amount of the available resources is needed for the setup of hardware and implementation of the required control mechanism, leaving only limited resources for the process development. Especially for small scale and prototype production with a high rate of changes, this is why fully automated solutions don't pay off and manual or partial manually assembly is preferred [1]. This paper introduces an approach how to separate the implementation effort of the hardware specific tasks from the process definition, allowing a fast and easy setup for new automation systems, due to simultaneous development and a high rate of reuse of previous solutions.

Keywords: software, framework, control system, one piece flow, assembly

Introduction

For most R&D orientated Enterprises as well as high tech production sites the implementation of automated processes is dominated by fast changing equipment and complex production or test setups. To be able to cover various automation tasks, a wide range of equipment (e.g. Sensors, Robots, Actuators) has to be integrated within the assembly system [2]. In order to make different types of hardware work together, engineering the control software usually takes most of the time. Even when components have been used in automated setups before, the interfaces have to be adapted to each other and the particular process requirements. As a result, a lot of work is required to integrate the same hardware again and again just because the setup of the hardware configuration has changed.

Even the control programs for the realization of the processes are developed specific to the particular application without the option to efficiently prepare for a future reuse of software code due to the interdependences of process and hardware. According to the experience with hardware setup and process development the effort needed for automation was about 70 % of the overall time leaving only 30% of the resources for the actual task, the process engineering.

To be able to focus on the process engineering it is necessary to decouple hardware setup and process development, at least in regard to the software programming. A

new approach within software for the automation of processes is required. This software system must enable a full reuse of previously developed solutions of hardware specific code as well as code describing the process flow. Once developed and implemented, a process like gripping a part in a special way or moving a robot towards a position and orientation, should be a procedure that does not require additional programming when used in another context. The same applies for the code providing hardware specific functions, like taking care about signal generation, positioning processes and communication with controllers.

The aim is to program the hardware typical functions within “hardware specific” and “process specific” code separated from each other. Thereby the point of view has to be changed form the conventional thinking “what the robot has to do in order to make the process happening” towards focusing on “what has to happen with the parts in order to assemble them”. Thereby, it doesn’t matter which kind of hardware configuration will be used to implement the process description. In addition, the software has to be extendable to allow a fast and easy integration of new components.

Based on this vision, a software system was designed and realized, achieving the separation of process related and hardware related code elements. Therefore the control system has to be grouped in different divisions representing characteristic aspects of the overall system:

- **Hardware representation:** Encapsulating all properties and functions of the available hardware, including data communication and control of hardware internal functions
- **Process representation:** All process specific elements have to be encapsulated within a code division, so that reuse is independent form the hardware components which are utilized for execution.
- **Basic system functionality:** The interaction between the hardware specific code elements and the process specific code has to be organized by a component. This component has to ensure that the process description will be put into action as adequate hardware is available.

Main condition is, that these software divisions are independent from each other although they need to interact in order to perform the designated tasks. The key benefit of this approach is the complete separation of the hardware control programming effort from the process description programming effort. In doing so, it is possible to setup and configure the processes without the need of knowledge about the hardware of the system that performs the later actions. Furthermore, the hardware control can be configured without the knowledge of the processes allowing a separated development process.

This is a key feature for a fast and efficient process development as existing solutions like gripping a component or aligning a component based on a reference signal can be reused without code implementation.

Application Scenario AutoMiPro

To realize the development and implementation of a software system with the abilities and specification described above an adequate application scenario is

required. Inside the AutoMiPro research project at the Fraunhofer IPT, logistics in terms of the design of hard- and software solutions for a changeable production respectively a one piece material flow within automated precision and micro assembly processes was developed. A demonstrator was rigged at the Fraunhofer IPT, consisting of four main components shown in figure 1. The skills of component processing are demonstrated within a diode laser assembly process containing various micro optical components. Therefore, hardware is needed capable of handling parts with different shapes, materials and handling requirements. As micro mechanical and optical parts of diode laser systems differ in their parameters due to the production process, flexibility is required. In order to increase efficiency by minimizing the use of sensor systems, every part must be treated as a unique entity with its corresponding data sheet characterizing the part. Thus it is possible to allow for handling and positioning of each part based on its real properties within a distanced process. In addition, information generated during the production process can directly be stored part specific, enabling a full traceability of each production step. The assembly process of a diode laser system itself is very complex. Alignment of at least one,

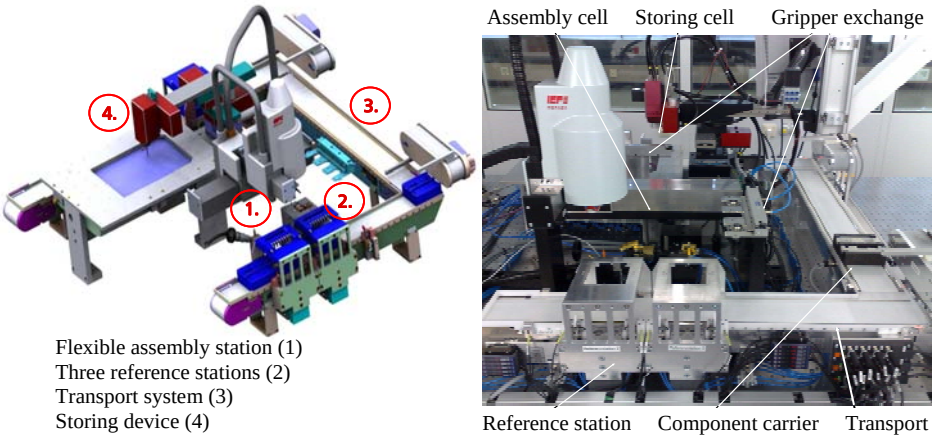

Fig. 1. Demonstrator setup of the AutoMiPro Project – Application: diode laser assembly

often more degrees of freedom is required for most of the optical parts. The alignment starts to get more complex as different alignment movements within the multiple degrees of freedom affect each other. By using the unique data for every part it gets possible to apply adjustment processes on different parts or at least part variations without change effort, once part properties are used to parameterize the alignment processes.

In order to allow for a company-overreaching production it gets even more important to separate the process description for hardware and form part specific properties and functions. As the production chain overreaches different hardware equipment and technology, a hardware independent description of the processing is required. This hardware independent process information is linked to parts itself, the same way the properties of the part are stored. The part so called “carries its own assembly description” through the production process. Assuming that all hardware control systems are capable of using this information, the necessary activation of hardware can be generated automatically inside the software. In this case no

additional input to the system is needed, realizing a clear separation between the hardware control issues and the product, respectively the process leading to the product.

Design of the Software System

To implement a software system that completely separates the programming of the hardware control form the specification of the process information two different types of software fields are needed. One for the hardware and its capabilities and one for the process information describing parts and procedures. The framework organisation element has to be capable to separate and organize the interaction between the hardware and the process description. In addition it must allow for flexible, fast and easy reconfiguration of the current hardware setup. Those three components, represent all aspects of a fully functional assembly system. (ref. figure 1).

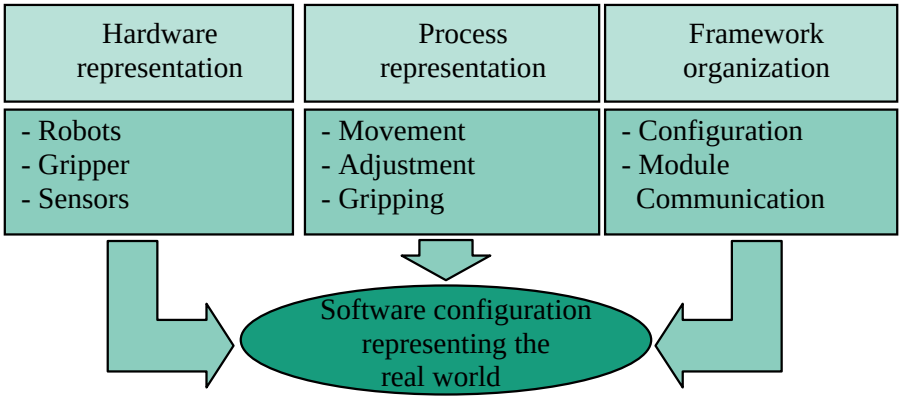

Fig. 2. Software components required to represent all aspects of an automated process flow including hardware control

To equip an organisation component with an appropriate logic, allowing to organize module interaction, causes additional effort for configuration. It also handicaps the goal of being universally usable and scalable as configuration specific, application related code is required [3]. Due to a required reduction of complexity it is essential, that the organisation component do not have to be aware of the specific system constellation in order to avoid the described effort for an omniscient organisation component.

Hardware representing Modules

The main goal regarding efficiency within the setup of an automated device is reusability of known and already realized functions, like accessing hardware via external interfaces or using positioning strategies for a robot system. Further, being able to apply changes within a running system without retesting and modifying further system programming is important to extent and optimizes functionalities.

Conventional control systems, often decentralized on robot controllers are highly integrated due to the limited abilities of the programming language or fundamental execution of NC-Code. All hardware components therefore have to be controlled by one device allowing for accessing all data and being able to integrate flexibility. In order to be able to use knowledge and skills of e.g. computer science personal, a PC platform and a common high level language like C#, capable to realize object orientated software has to be utilized.

The object orientated programming allows for separation of properties and functionalities into separate software objects, each representing the smallest entity of a hardware component. A device with an integrated pneumatic actuator for example would be separated into the device specific part and the pneumatic actuator. This actuator itself is connected to a standard component, a pneumatic valve, which is already available and completely represented within the software framework. Thereby the functions for switching valves and accounting for actor properties, already is included within the software and can be reused.

In order to minimize the effort of extending the system with additional objects the method of inheritance of software objects is used. Inside the developed software structure every part of the real world is represented by a single software object, containing all information and providing all functions of its real-world counterpart.

By strictly separating and isolating functions and properties into software objects, it gets possible to modify and even replace whole elements with no or at least minimal effort. Communication between these objects is realized by primitive commands respectively messages. Therefore each object is programmed to be capable to process certain messages and commands.

In order to avoid the implementation and configuration of an omniscient organisation component, the software objects, representing hardware components, are organized within a hierarchical, tree like structure. This structure between the software objects has to be defined at system start up based on the relations of the real system configuration. The system automatically keeps it up to date during automated processing. With this approach production hardware as well as product parts can be described the same way. Figure 3 illustrates how a gripper carries a part, in this case a **F**ast-**A**xis-**C**ollimation lens. The gripper is subordinated underneath the bottom side of the gripper interface, which is connected to the upper interface side. The upper interface is linked to the robot taking care about its positioning.

All information concerning the position and orientation of components is defined relative to the superior component. This allows for a fully automated update of the data, allowing automated configuration modifications like gripper exchange or the gripping and releasing of parts. Therefore the part is always dedicated to the component defining its position within the real setup. If it is placed within a fixture, its position is defined relative to the fixture. In case a robot respectively a gripper handles it, it is dedicated relative to the gripper, which defines its current position. In order to extract the information about position and orientation of a component, the part can be “asked” for its position. Once this request is started, internal functions all objects provide, causes the calculation of the absolute position based on all relative data independent to the current configuration of the system.

In order to ensure proper precision of the virtual model, certain process steps can be used to update the relative position of a component. A fully traceable production

can be realized linking all historical data and part information within the data set for the assembled product. Even relevant process data can be collected within processing and assigned to the processed components for full traceability within conventional production as well as for process ramp up or process development.

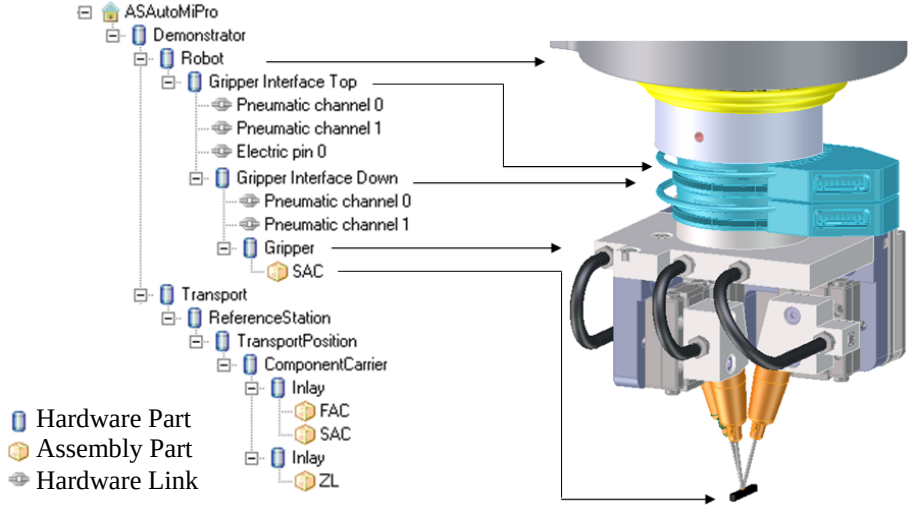

Fig. 3. Example for the representation of real world elements within the software system

In addition, the modular concept of the software control guarantees an easy exchange or modification of hardware components or process logics. If for example a new gripping device is added, only the gripper software object needs to be exchanged. Therefore, an existing gripper class definition can be derivated to a new gripper subclass or only be parameterized without adding further functions. All other participating modules, controlling the other hardware and processes, stay as they are, with the ability to use the new gripper right away. Thus a toolbox of different software modules is created. The operator is allowed to simply select and connect the required modules and configure the hardware setup as needed.

Process Representing Control Modules

The second software element addresses the way process information is described. Due to the modular and flexible setup of the assembly, the description of the processes has to be hardware independent. This is crucial for parallel development of hardware and process as well it is a basic condition for reusability of developed process steps. Therefore, all process descriptions are defined parameterized. Parameters can be software objects representing hardware components like grippers or representations of objects being processed as well as simple parameters. All absolute values, in particular position and orientation values, required for the execution of handling processes are thereby defined based on the component data, calculated on the fly once the process description is executed. This ensures that current configurations and positions of robots or other variegating properties are included once a command or process description is executed.

On behalf of a deterministic process development, detailed process descriptions can be used for defining the processing based on elementary commands, like “move left 2 mm”, with no margin for variances. Due to changeability demands and the need for simplification of the process development, more abstract commands like “position part A next to part B with an offset of 2 mm” can be executed and realized. For more complex logics, as alignment processes, a process command object is available allowing for encapsulation of process logic by the generation of command primitives, executable by the software framework.

Process Example – Diode Laser Assembly Process

To adjust a micro optical lens in front of the diode laser source, multiple different procedures have to be executed. As described, the information specifying these procedures are stored as a list of components dedicated to the part information of a so called master part, describing only the “what is to do” but not the “how to do it”. This includes command primitives and complex process descriptions like adjustment and alignment functions e.g. for the fully automated active alignment of a **F**ast-**A**xis-**C**ollimation lens. As the assembly process is started, for example the FAC lens is asked to move itself towards the diode laser in, in order to get it into adjustment position as shown in figure 5. For this operation it is completely regardless where the part is located or how it will be transported. The part, supposed to move itself just

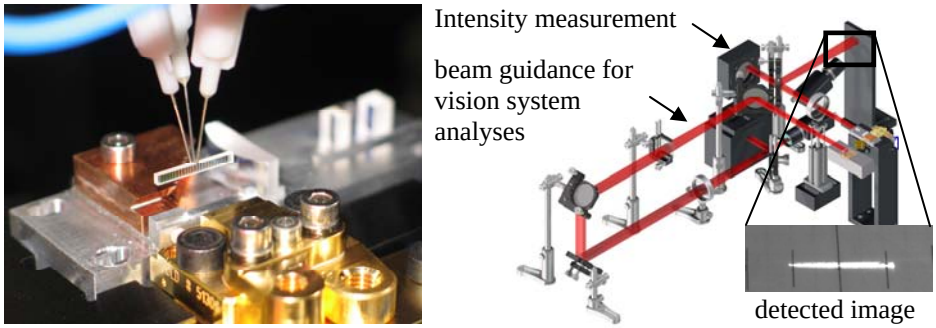

Fig. 4. Fully automated FAC alignment at the Fraunhofer IPT using hardware independent process descriptions and process independent hardware programming

creates a new command to provoke its transport to the target position, as it is not capable to move itself. This command is always addressed to the superior software object until a software object is identified, accepting the command for execution (ref. Figure 3). In case of the linking of a positioning task, each rejection of the command causes a modification in order to accomplish the initial positioning request. As the part wants to be moved and is not capable to move, it asks the gripper, currently defining its position to move. The gripper, itself not capable to move, recalculates the moving request in a way that it wishes to get moved by its superior object so that the FAC lens will be moved the way wanted to, initially. By this, a command is transferred through the hierarchical system, although no component is aware of the

overall structure, until a software object accepts execution of the current command. By this means, boundary conditions like acceptable gripping methods or gripping force limitations can be accounted for in an unknown environment of hardware components. This enabled for parallelisation of development and extendibility of already producing systems but also cause only partial predictable process realisation.

Although it is possible to assign commands to be executed by predefined hardware components, using their unique object ID, instead of using hardware independent commands. This option keeps the concept scalable for development and introduction within industrial applications; even some of the advantages of the concept thereby get reduced. Similar to parts provoking their own processing by creating command primitives, process objects itself can create requests to get for example information out of a vision system as it is required for the handling and alignment of a FAC lens as shown in figure 4.

Summary and Outlook

The presented software framework is currently in implementation. Defining hardware-independent positioning processes and executing theses on different hardware configurations is already put into application. Further expansion of the system by integration of additional devices is ongoing. As the designed system provides a solid base for complex automation tasks, additional research will be conducted within the field of the design of more complex process objects in order to allow for automated rigging and adaptive or self learning processes.

Acknowledgements

The authors thank the German Federal Ministry of Education and Research (BMBF) within the project »AutoMiPro«, part of the Framework Concept »Research for Tomorrow’s Production« (02PB3140) managed by the Project Management Agency Forschungszentrum Karlsruhe, Production and Manufacturing Technologies Division (PTKA-PFT).

References

1. J. Hesselbach et al.: mikroPRO – Untersuchung zum internationalen Stand der Mikroproduktionstechnik. Essen: Vulkan-Verlag, 2002
2. Robot Visions to 2020 and beyond – The Strategic Research Agenda for robotics in Europe, 2009
3. J. Bara, L. Cararinha-Maos: Coalitions of manufacturing components for shop floor agility. International journal of networking and virtual organisations IJNVO 2003 ISBN 1470-9503