High-level integration of components into a robot cell using semantically annotated state chart descriptions

Dipl.-Ing. Martin Naumann, Univ.-Prof. Dr.-Ing. Dr. h. c. Alexander Verl
Fraunhofer Institute for Manufacturing Engineering and Automation IPA, Stuttgart, Germany

Summary / Abstract

This paper describes a concept for the reconfiguration and programming of a robot system for different application types by a non-expert end-user. To achieve this, it is necessary to exchange components on demand and at the same time reuse available process definitions, offer a high-level programming interface for the end-user and generate executable code for the involved components automatically. The presented concept is based on component, process and application descriptions realized as semantically annotated state charts. Matching of process and component descriptions determines executable processes. Fusion of process, component and application description results in executable code. The concept is intended to provide a basis for the realization of robot assistants for small series production where a robot system must be a flexible to use and easy to program tool that supports workers in their daily business.

1 Introduction

Industrial robots are widely used in mass series production [1]. The most prominent example is the body-in-white manufacturing in the automotive industry. In this scenario, robots are set up in a very structured environment for a specific task once and then repeat this same task for a very long time, typically several years. Available industrial robots are optimized for exactly this scenario where high performance is the determining factor for the economic success of a robot installation. The costs for setting up the robot are marginal compared to the long time the robot is operating afterwards.

In small series productions, the prerequisites are completely different: tasks are changing regularly, the environment is much less structured and full automation is seldom possible [2]. To apply robot systems in small series production a paradigm shift is necessary: the time and effort necessary to set up a robot system for a specific production task may no longer be neglected because it is the determining factor for economic success. This will result in robot systems that are optimized for flexibility, usability and performance instead of just performance.

This paper describes an approach to gain this flexibility by semantic integration of the components of a robot system with the goal to allow a non-expert user of a robot cell to set up new applications.

2 State of the art

The state of the art for setting up an industrial robot system involves the following steps:

1. Mechanical integration of all components
2. Configuration of the individual components to set up communication
3. Application specific programming
    a. of cell control PLC program
    b. of robot motions (offline/online)
    c. of other software components
    d. of user interface for end-user
4. Test, Parameter tuning, delivery to end-user

All steps are performed by robot experts with the help of dedicated and complex to use software tools like e.g. STEP7, TwinCAT or Delmia-V5. Only big companies have these robot experts in-house. All other companies must rely on support of external experts, typically from system integrators. In all cases, the set up of a robot system using the conventional approach takes a lot of time and costs and therefore makes sense only for mass series production.

In case of a changing production task different cases can be distinguished:

- **Case 1:** Parameter adaptation because of e.g. new product variant → involves step 4
- **Case 2:** Changes in application specific program because of e.g. new product → involves step 3, 4
- **Case 3:** Components need to be exchanged for a new type of application → involves all steps

Case 1 can be performed by the end-user already today if an appropriate user interface is provided.

For *Case 2* application specific solutions exist that allow an end-user to cope with the complexity by offering an intuitive and flexible user interface. One example for welding applications is the Inteach-solution developed at Fraunhofer IPA within SMErobot [3]. This solution allows a welding expert (but not necessary a robot expert) to define welding applications by dragging the robot, giving speech commands and manipulating the robot movements on a touch panel showing a 3-D-model of the system. Figure 1 shows how the system is used.

Figure 1 Inteach system for intuitive programming of welding applications

This example works well for one specific application (welding) using the set of components that were foreseen when developing the system.
In small series production, the capacity utilization of a robot system is often not high enough to allow amortisation in an acceptable time if the system is restricted to one type of application. Therefore, it would be desirable to use a robot alternatively for e.g. welding, pick and place and machine tending applications. To achieve this, it is required to reconfigure the robot system by adding and removing components like tools, sensors etc.. This scenario resembles the above described *Case 3*.
Up to now, no solutions exist that allow end-users to reconfigure a robot system in order to use the system for different types of applications. The proposed semantic integration of components into a robot system is an approach to realize such systems.

3 Approach

This chapter describes the chosen approach for setting up a robot system for small series production. The chapter starts with the description of the desired workflow when setting up an application, continues with the corresponding role model, the control architecture, reusability requirements, the high-level programming interface and ends with an overview of the resulting robot system.

3.1 Desired workflow

To make use of robots in small series production a different approach is required. The above mentioned steps need to be performed either automatically or in a way that a non-expert end-user of a robot system is able to perform them. This means:
Mechanical integration of the components may not involve any mechanical design effort. Instead, standard interfaces e. g. to change the robot tool must be used that provide a standard mechanical interface and standardized plugs for power (electric, air) and communication connections. This topic is not the scope of this paper.
Setting up of communication to the components of the robot system may not involve any user-interaction. Therefore, communication protocols must be used that allow plug-and-use, meaning that configuration of the communication is done automatically. Examples for such protocols are XIRP and UPnP [4, 5]. This topic is not the scope of this paper.
Application specific programming: this step requires interaction with the end-user because he/she needs to supply information about what should be done. The end-user should be able to supply this information in an as intuitive way as possible by specifying the sequence of high-level-steps of an application and parameterizing each step. In no case, the user should be required to supply device-specific information or source-code of whatever kind. Generation of an executable program out of this high-level application description must be done automatically. This is the topic of this paper.
For **testing and parameter optimization** simulation in combination with 3-D visualization should be used to allow the user to find as many errors as possible before testing the application on the real system. To optimize the application on the real system intuitive means to interact with the robot must be available, e. g. guiding the robot by hand. This topic is not the scope of this paper.

This workflow imposes new requirements. The most important ones are automatic set up of communication to all devices, high-level programming and automatic generation of an executable program under the boundary condition of the open world assumption meaning that there is no predefined set of devices and operations. Therefore, solutions are necessary that allow the integration of new devices, the definition of new operations, the presentation of the resulting capabilities in a high-level programming environment and finally the generation of executable code with the open world assumption in mind. To fulfil these requirements the state of the art situation where almost all knowledge relevant for setting up a robot system is only available in the head of the persons actually setting up the system or in non-machine-readable text documents like handbooks or specifications is not sufficient. Instead, this knowledge must be made explicit.

3.2 Role model

There are three types of knowledge involved when setting up a new application: knowledge about the specific application requirements/products that should be manipulated; knowledge about the components/resources of the robot system and knowledge about specific, but reusable processes that need to be executed to achieve the desired result when manipulating the products with the available resources [6]. The knowledge about components/resources of the robot system and the knowledge about specific processes can be generated offline by component-experts and process-experts because it is not application specific. The knowledge must be available in machine-readable

form to be imported into the robot cell when needed. In contrast, the application specific information must be supplied by the end-user online when setting up the application. All three types of knowledge must be joined to get an executable program. This fusion of the knowledge is called code generation and must be done automatically because it exceeds the capabilities of the end-user by far. The described role model is illustrated in figure 2.

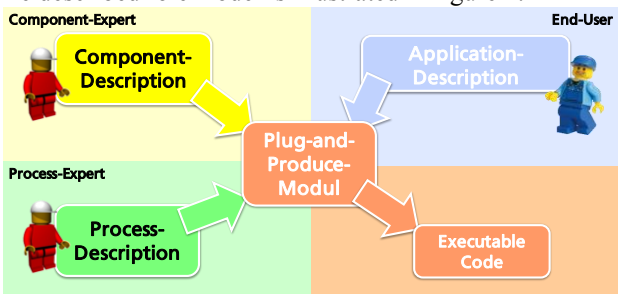

Figure 2 Intended role model

3.3 Control system architecture

To achieve the above described workflow and role model a modular system architecture is required that fulfils the need for reusability and loose coupling imposed by the above described workflow. A robot system consisting of several components has many similarities with a distributed system in computer science. Therefore, architecture concepts developed for distributed systems can also be applied in the robotics domain. Especially interesting are Distributed Object Architectures (DOA), Component Based Architectures (CBA) and Service-Oriented Architectures (SOA). Comparisons of the three architectures (see figure 3) reveal that all three architectures support reusability and extendibility, but a service-oriented architecture is preferable if loose coupling is required [7].

Criteria	DOM	CBA	SOA
Coupling	tight	medium	loose
Reusability	possible	possible	possible
Extendibility	high	medium	high

Figure 3 Comparison of DOM, CBA and SOA according to [7]

The application of the SOA principles to robot systems can be interpreted such that the components of a robot system offer one or more services like e. g. a gripper offers the `fix_object` service. To use the services offered by the devices the functionality and also the interface to use the service must be described in a machine-understandable way. All components must be connected via a network to a central cell controller using communication protocols that allow automatic discovery and setting up of communication. Part of this cell controller is an execution engine that orchestrates the services offered by the components based on an application description.

3.4 Reusability

There are different aspects of reusability that the described approach addresses.

3.4.1 Reusability of the control system itself

The resulting robot control system should be usable for many applications requiring many different components. The scope of applications and components the system should address cannot and should not be defined and thereby limited while designing the system. Therefore, it is necessary to keep all component and application specific knowledge out of the robot control system and its implementation. Instead, application and component specific information should be integrated by means of loadable descriptions that contain the specific information.

3.4.2 Reusability of application information

Application information should be reusable in two scenarios: in, at least partly, similar applications and when using different components with similar functionality.
An application is a synchronized sequence of parameterized services offered by components as illustrated by the upper part of figure 4. Sometimes, parts of an application can be reused in other applications as depicted in the lower part of figure 4 thereby saving time to define a new application compared to defining this application out of services. Therefore, it is required to allow for the possibility to group services with different granularity. These grouped services are called processes and their descriptions are called process descriptions in this paper. Processes are the basic building blocks for the user to define a specific application.

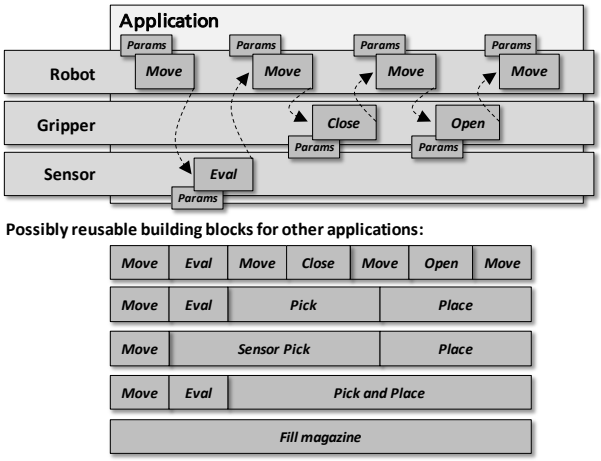

Figure 4 Reusability of parts of an application

If such a process is reused in a different application, it is likely, or at least possible, that a different set of components is used for this application, e. g. a robot from another manufacturer and an electric gripper instead of a pneumatic gripper. It is very unlikely that these different components offer exactly the same interface, but they offer more or less the same functionality. To make a process description reusable in such a scenario an abstraction

mechanism is required. This abstraction mechanism must be able to cope with different communication protocols and with different component interfaces. Therefore, a two-layered approach is chosen. Communication drivers and a communication abstraction layer are used to integrate different communication protocols into the system while the description of the functionality of a component (see chapter 5 and 6 for details) allows hiding specific interface properties. Figure 5 illustrates the two-layered abstraction mechanism.

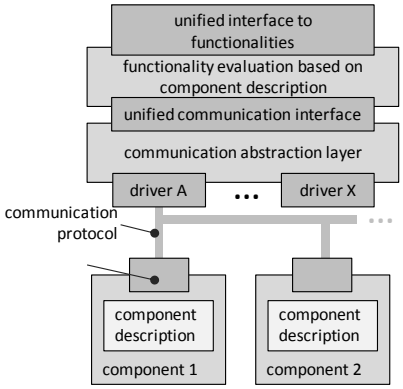

Figure 5 Abstraction mechanism for communication protocol and component interface specific properties

3.5 High-level programming interface

[8] describes different levels of abstraction for robot programming. These are described in figure 6.

Product	The resulting product is described. The system plans the operations accordingly.
Process	A sequence of known operations and their parameters are described.
Tool	The movement of the tools is described. The user knows the process.
Arm	The movement of the arm is described. The user knows about the used tools.
Joint	For each position the joint values are described

Figure 6 Robot programming levels of abstraction

Programming on process level seems to be appropriate for the approach described in this paper because the user is very familiar with the process and the parameters to use and therefore should be able to supply the required information. Technically, it would be possible to build a planner on top of the described system that allows programming on product level and results in a process sequence. But this is not the scope of this approach.

3.6 Resulting robot system

The resulting robot system consists of components that offer their functionality via services to a central cell controller. The automatic integration of the components into the cell controller is managed by specific drivers for different communication protocols. Nevertheless, a unified communication interface on top of a communication abstraction layer allows communication with the components without the need to take care of protocol specific properties.

A high-level programming interface allows the user to define an application on process level using predefined building blocks, called processes. It is therefore the job of the user to define the sequence of processes and their parameters. The available predefined processes are stored in a process library.

A plug-and-produce module serves as hub between components, processes and application description. It collects component descriptions and process descriptions, evaluates these descriptions to determine processes that can be executed by the current set of devices and propagates these processes to the high-level programming interface. The user then defines an application using these processes. It is then the job of the plug-and-produce module to merge component descriptions, process descriptions and the application description into executable code. An execution engine is then able to run the application by triggering the services offered by the components accordingly.

Figure 7 gives an – not complete – overview of the resulting robot system.

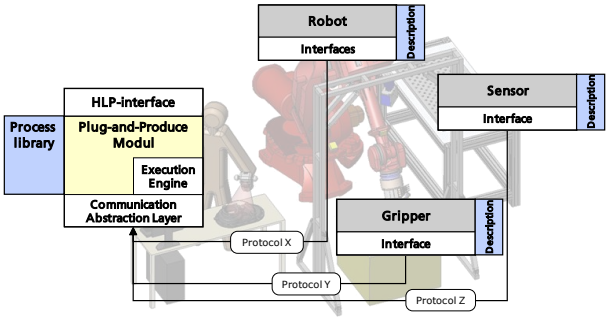

Figure 7 Block diagram of the resulting robot system

The plug-and-produce module together with the mentioned descriptions are the key elements that supply the functions that allow an end-user to set up a robot system for a specific application without expert knowledge about devices and processes.

4 Semantic integration concept

Semantic integration in this context means the integration of components into a system such that the functionality of these components is known by the system. To achieve this, the plug-and-produce module must be able to do two things: First, it must be able to match the described functionality of the available components with the functionality required by the processes stored in the process library to determine the processes executable by the available set of components. These executable processes are the input to the high-level programming interface. Based on these processes the user generates a specific application description. Second, the plug-and-produce module must be able to fuse this application description, the component descriptions and process descriptions to automatically

generate an executable program. Therefore, the following information must be provided by the descriptions.

- **Component description:** has to provide information about the functionality of a component and how this functionality can be accessed through its interface.
- **Process description:** has to provide information about the functionality required for the process, the sequence of actions when executing the process and the parameters that can be tuned to adapt the process to a specific task.
- **Application description:** contains a sequence of processes and values for the parameters of each process. This information is provided by the end-user.

4.1 Information representation

Harel's state charts [9] with semantically annotated states seem to be a good possibility to represent at least part of the information of these descriptions. They can be used to describe the internal control model of components using states that represent real world situations. Transitions describe either input commands/data required to trigger a state change or output commands/data that result in a state change triggered by component-internal events. Figure 8 shows the example state chart of a gripper with an additional turning axis and a sensor to detect gripped parts.

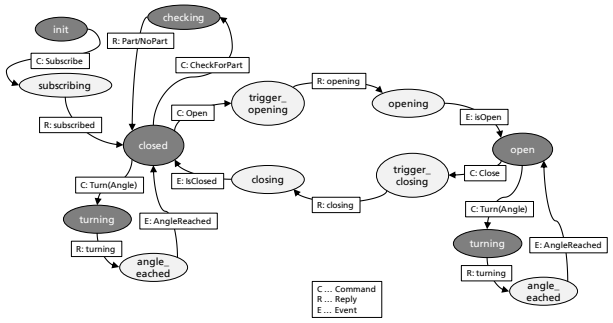

Figure 8 Component state chart of internal control model for a gripper; semantically annotated states are dark grey, internal states of the component are light grey; transitions describe input/output data or commands

Figure 8 shows two types of states. States that represent a real world situation like e. g. `closed` or `open` and states that represent component-specific internal control modes like e. g. `trigger_opening` or `subscribing`. States that represent a real world situation have a defined meaning and are therefore called semantically annotated states.
These states have two purposes. Because each of these states has a specific meaning and is grounded to a specific real world situation, these states can be used to describe the functionality of a component. The set of all semantically annotated states contained in a Component State Chart describes the offered functionality of this component.

Second, they can be used to relate component descriptions and process descriptions. Process descriptions can be modelled using the possibility of Harel's state charts to describe parallel actions. Each component involved in a process is described as one sequence of states (when taking into account only the nominal case). Interactions between components like synchronization of components can be described with the help of additional states. Figure 9 shows the example of a pick process. Because process descriptions are not component-specific, they contain only semantically annotated states and the transitions do not contain input/output data or commands.

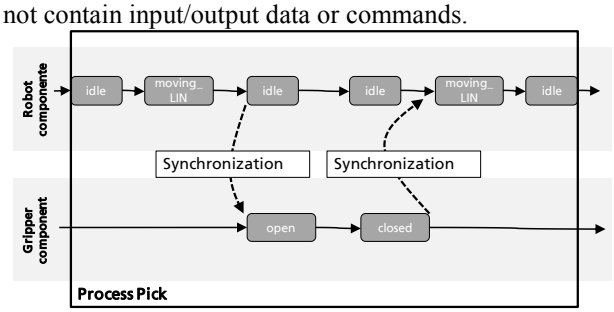

Figure 9 Process state chart describing a pick process involving a robot and a gripper

Semantically annotated state charts can be classified in a hierarchy according to their functionality. The top level of the hierarchy contains states describing general functionalities like e. g. `Move`. Similar to the class-subclass concept in object oriented programming, more specific functionalities like e. g. `Move_Straight` are then subfunctionalities. This classification allows determining processes that are executable by checking if all semantically annotated states of a process state chart are supplied by the set of available components. If yes, an executable state chart can be generated and the process can be executed.

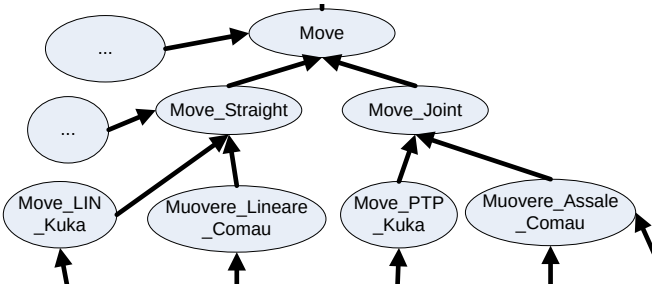

Figure 10 Example for the classification of functionalities

4.2 Information fusion

This chapter describes the algorithm to fuse component descriptions, process descriptions and application description to gain executable code. Figure 10 gives an overview of the workflow.
The fusion of component description and process description is based on a graph search algorithm like e. g. the

Dijkstra algorithm. For each transition between two semantically annotated states in a process state chart a path in the Component State Chart of the used component must be found. This path contains also states that are related to the internal control model of a component and transitions that describe the necessary communication to trigger the state changes. Therefore, using the resulting state chart the process can be executed with the available devices. Therefore, this state chart is called Executable State Chart.

The fusion of application description with the resulting Executable State Charts is more or less a concatenation of the latter taking into account certain boundary conditions. Figure 11 gives an overview of the workflow.

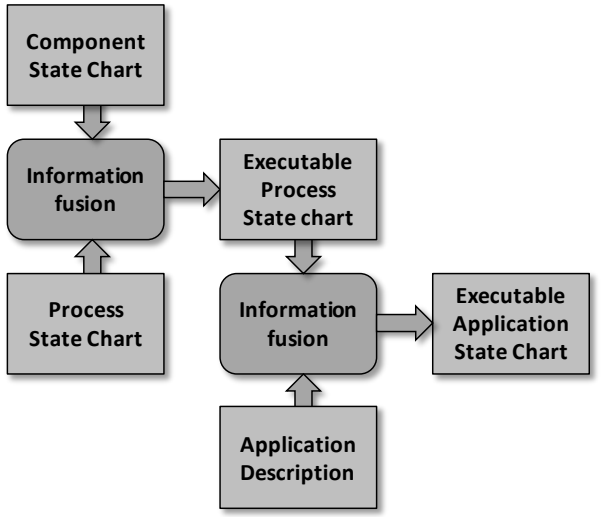

Figure 11 Workflow to generate an Executable State Chart

5 Conclusion

The presented concept offers the following advantages:

- component, process and application domain knowledge is explicitly described,
- experts in their field generate the respective descriptions,
- process descriptions can be reused for different components (with similar functionality),
- component descriptions can be reused for different processes (requiring similar functionality),
- application descriptions are defined on a high-level of abstraction. Therefore, an non-expert end-user can define an application,
- the high-level programming interface adapts to the functionality provided by the connected components and offers only executable processes to the end user,
- code generation is done automatically by fusing process, component and application descriptions.

A first evaluation in a bin picking application showed promising results. Further evaluations of the concept in other example applications will show the performance and usability of the concept for different applications. The author is optimistic that the results will be promising, especially taking into account that the concept is aimed at rather simple applications like pick and place, machine tending and simple assembly.

6 Literature

[1] IFR statistical department: World Robotics 2011.

[2] Hägele, Martin: Interaktiv und kostengünstig: Eine neue Robotergeneration für kleine und mittelständische Fertigungen. In: Automation & Drives (2008) 52, S. 12-14.

[3] Meyer, Christian; Hollmann, Rebecca; Parlitz, Christopher; Hägele, Martin: Programmieren durch Vormachen für Assistenzsysteme – Schweiß- und Klebebahnen intuitiv programmieren. In: it – Information Technology 49 (2007) 4, S. 238-246.

[4] VDMA Einheitsblatt 66430-1: XML-basiertes Kommunikationsprotokoll für Industrieroboter und prozessorgestützte Peripheriegeräte (XIRP) – Teil 1: Allgemeine Vereinbarungen.

[5] ISO/IEC 29341-1:2011: UPnP Device Architecture – Part 1: UPnP Device Architecture Version 1.0

[6] Schleipen, M.; Drath, R.: Three-view-concept for modeling process or manufacturing plants with AutomationML. In: Proceedings of the 14th IEEE International Conference on Emerging Technologies and Factory Automation 22 - 26 September 2009, Palma de Mallorca, Piscataway, NJ: IEEE, 2009.

[7] Amoretti, Michele; Reggiani, Monica: Architectural paradigms for robotics applications. In: Advanced Engineering Informatics, Volume 24, Issue 1, January 2010, Pages 4-13, ISSN 1474-0346, DOI: 10.1016/j.aei.2009.08.004.

[8] Hägele, M.; Nilsson, K.; Pires, J. N.: Industrial Robotics. In: Siciliano, B.; Khatib, O. (Ed.): Springer Handbook of Robotics. Berlin, Heidelberg: Springer 2008.

[9] Harel, David; Naamad, Amnon: The STATEMATE semantics of state charts. In: ACM Trans. Softw. Eng. Methodol. 5 (October 1996) 4, pp.293-333.