A Flexible Architecture for Automatically Generating Robot Applications based on Expert Knowledge

Miroslav Macho, Ludwig Nägele, Dr. Alwin Hoffmann, Dr. Andreas Angerer, Prof. Dr. Wolfgang Reif
Institute for Software & Systems Engineering, University of Augsburg, 86159 Augsburg, Germany
Email: macho@isse.de, Phone: +49 821 598-2227, Fax: +49 821 598-2175

Abstract

In this paper, we propose a general software architecture for off-line programming platforms to semi-automatically program a robot system and its end-effectors in industrial automation. It considers that such platforms should be geared towards domain experts – especially when regarding low-batch size manufacturing of customized products as part of *Industry 4.0*. Hence, it supports a process-oriented guidance for complex manufacturing tasks and includes the possibility for interactive planning with domain-specific user interfaces. Moreover, the extensibility for new domains, work-flows, algorithms, user interfaces or hardware plays an important role in the proposed approach. Finally, the architecture was successfully evaluated on an off-line programming platform for the manufacturing of carbon-fibre-reinforced polymers (CFRP) which can be characterized as a low-batch size production process.

1 Introduction

The International Federation of Robotics (IFR) estimated the total worldwide stock of operational industrial robots at approximately 1.5 million units to the end of 2014 [1]. According to this statistics, the most important sector for industrial robots is the automotive industry followed by the electrical and electronics industry. Both industries have in common that they widely use industrial robots for mass production. Due to the tedious programming of industrial robots [2], robotics systems still are not often used in production processes with high variability or low-batch sizes. However, customers are more and more expecting highly customized products today. The vision of *Industry 4.0* with its smart factories as well as interconnected intelligent machines and products supports this demand for variable and customized products. Manipulators – either as a cooperative team or in combination with a mobile platform – are highly flexible and freely programmable machines which are appropriate for *Industry 4.0*. Hence, new ways and means for programming industrial robotics systems for customized, low-batch size production are required.

For that reason, we have inspected a production process that differs from a mass production, i.e. the manufacturing of composite components made of carbon-fibre-reinforced polymers (CFRP). In collaboration with the German Aerospace Centre (DLR), we have developed a offline programming platform tailored to the special requirements of the CFRP production [3]. CFRP components can be composed of several hundreds or thousands of textile cut-outs. During the CFRP production process, each cut-piece must be picked from a storage and placed into a mold. In this way, hundreds of cut-pieces are layered to form of the product's final shape. In an automated production, this leads to complex robot movements and end-effector actions that repeat for each cut-piece, however, with slight *differences* depending e.g. on the size, shape, or material of the cut-piece. For a single CFRP component, the programming of a complete production process with standard programming tools and algorithms can be very tedious [4]. Furthermore, the automation of this process is ongoing research and, thus, the work-flow can evolve or even completely change and novel handling end-effectors can be developed requiring new programming approaches.

Generally, there are production processes like the CFRP manufacturing which consist of a large number of *similar* production steps that may slightly differ in robot movements or end-effector control depending on various parameters. Some of these parameters can be *computed*, e.g. based on geometric information included in CAD models. Some parameters cannot be computed without extensive effort, because they are optimization problems (e.g. optimal actuator pose for placing a cut-piece into a mold). Other parameters are based on *domain expert knowledge* and, thus, depend on user input (e.g. maximal robot movement speed during transfer movements in order that cut-pieces hold on the end-effector).

This change in production processes – especially due to the *Industry 4.0* efforts – raises special demands regarding the programming of robot systems and end-effectors. For the automation of such production processes with a reasonable effort, a computer-assisted off-line programming [5, 6] approach is required. Here, robot programming does not take place directly at the robot cell, but in a simulation environment with a virtual representation of the robot cell. There are several advantages that make off-line programming outstanding for introducing and planning new manufacturing processes (e.g. the robot cell is not occupied during programming, additional information can be visualized, and no collisions can occur).

According to a previously conducted analysis [3], a main requirement for such a programming approach is that it should be geared towards domain experts. Hence, it should lead the (expert) user through the process work-

flow, generate robot movements as well as end-effector actions automatically and combine them with additional user input, if some process steps need domain expert knowledge or are to complex to be computed. As user input can be required during or improve the work-flow generation, we call this approach *semi-automatic programming*. It also allows domain experts to experiment with new parameters for a specific production step while keeping other parameters on default settings. To observe the impact of a parameter change, it should be possible to adjust these parameters in an iterative way and to visualize the defined process or selected production steps.
Besides the semi-automatic program generation with expert knowledge, extensibility is a further key requirement for the software architecture: It should be extensible with new work-flows, algorithms, domain-specific user interfaces and hardware (i.e. robots or handling tools). To support this extensibility, the software architecture has to be modular enabling a dynamic interconnection of individual modules. Moreover, it should be possible to integrate modules for computation as well as modules for domain-specific user interaction to involve the domain expert and his knowledge.
Hence, we propose in this paper a general software architecture for off-line programming platforms to semi-automatically program a robot-based manufacturing processes. Being a modular approach as mentioned above, we introduce and explain so-called *Task Contribution Units* as base modules for the approach in Section 2. Subsequently, the successful application of this approach to CFRP manufacturing is described in Section 3. While a a short overview of related work regarding robot programming is given in Section 4, the paper is concluded with Section 5.

2 Task Contribution Units

Software architectures aimed for creating robot-based process definitions often deal with challenges regarding extensibility, reuse, time-effort, collaboration and complexity problems. This is especially the case when different domains are targeted within one application, such as hardware, workpiece-specific process information, mathematical algorithms, etc., and several experts from different domains have to work together. Quite often, also huge numbers of workpieces or process-steps make coordinated collaboration very difficult. As a solution to these challenges, we propose a modular and flexible architecture based on *Task Contribution Units* (*TcUnits*), that enable a structured, separated development of domain-specific parts contributing to one mutual result. For this, the manufacturing process needs to be split into small partial tasks which describe individual steps of the process. A process *'mounting a car door'*, for example, might be composed of tasks for picking-up, transporting and attaching the door. These tasks, of course, can be highly dependent on other tasks and their concrete parameters; In order to implement such a task, lots of additional information and calculation is needed (i.e. *Task Contributions*), e.g. mathematical and geometrical calculations and optimizations, information about robots and the environment, data look-ups, depending tasks, and also situational information given by domain experts.

2.1 Definition

In the proposed architecture, the common interface of all these atomic contributions is called *TcUnit*. While defining a specific output data type (i.e. *result type*), a *TcUnit* encapsulates its internal computational mechanism for retrieving the concrete *result* of that respective type. We identified three different kinds of *TcUnits*:

1. **Data provider *TcUnit***: Most production processes rely on CAD data or manufacturing information, which is to be obtained from files, databases, etc.. This kind of *TcUnits* loads and provides such data for generation processes.

2. **Computational *TcUnit***: Usually, a manufacturing process is defined by lots of computational values, such as geometric relations, best fit parameters, or derived setting values. Computational *TcUnits* provide such results by using mathematical mechanisms or algorithms.

3. ***TcUnit* basing on expert input**: Some information of manufacturing processes are conditioned situational and can not be computed automatically. Hereby, investigation of an expert is needed who contributes additional information to the process. Such *TcUnits* are intended to involve expert knowledge into the process definition where necessary (see 2.3).

Regardless of its kind, each *TcUnit* can specify an individual set of input parameters (i.e. *input types*) which need to be provided in form of concrete inputs in order to generate its result. Assume the notation of a *TcUnit* given its result and input types as follows:

$$TcUnit : RESULT \Leftarrow INPUT_1, INPUT_2, ... \quad (1)$$

A *TcUnit* providing a *robot's target position* for mounting a car door, for example, might require several concrete inputs: geometrical information of the car door, relative position of the car body, robot and tool to be used for performing the mounting.
Assume a set of $TcUnits_i$ with their respective result types **A** through **E**. Their given input types establish dependencies among each other:

$$\begin{aligned} &TcUnit_1 : A \Leftarrow B, C && TcUnit_4 : D \Leftarrow E \\ &TcUnit_2 : B \Leftarrow C && TcUnit_5 : E \Leftarrow \\ &TcUnit_3 : C \Leftarrow E \end{aligned} \quad (2)$$

All *TcUnits* together form the ecosystem of available *TcUnits*, and their result types denote the set of types which can be generated by that ecosystem (i.e. *generatable types*). **Figure 1** demonstrates the *TcUnits* and their generatable types of an ecosystem along with their dependencies (i.e. *dependency graph*). *TcUnits* are connected

by arrows to their respective input types. Edges are used to denote the relationship between *TcUnits* and their result types, at which one type can have edges to multiple *TcUnits*. That means, a result of that result type can be generated alternatively by all connected *TcUnits*. Cycles are not allowed in this dependency graph. However, they are not supposed to happen since it does not make any sense that two results depend on each other in order to being generated.

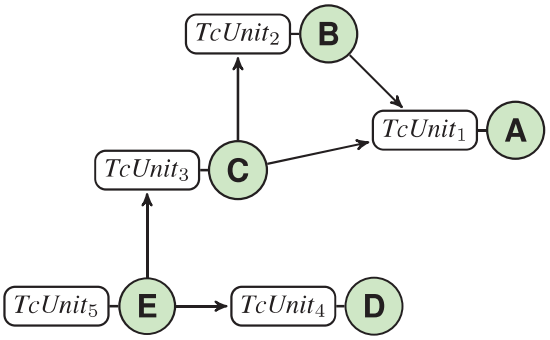

Figure 1: Example ecosystem of *TcUnits* with its generatable types and dependencies.

When generating result **A** by $TcUnit_1$ inputs **B** and **C** are required to be generated before, and finally **E** needs to be the very first being generated by $TcUnit_5$. Generally, the generation recipe for a certain result type is expressed by a sub-tree of the dependency graph (see example in **Figure 1**), consisting of necessary *TcUnits* and their dependencies respectively. In the given example (where **A** should be generated) it is called *candidate tree* for generating **A**. In this case, result type **D** and thus $TcUnit_4$ are irrelevant and not part of the candidate tree. When topologically sorting such a candidate tree, a serialized order of *TcUnits* arises which is also called generation trace. In the given example, the one possible generation trace for generating **A** is:

$$Trace_1(A) : TcUnit_5 \rightarrow TcUnit_3 \rightarrow TcUnit_2 \rightarrow TcUnit_1 \tag{3}$$

When generating results of certain types by following the order of these topological traces, it is always ensured that all input values needed by the respective *TcUnit* have already been generated before.

2.2 Extensibility

When extending the ecosystem with new *TcUnits*, the dependency graph will get additional branches depending on the respective result type and needed input types of the *TcUnit*. A more branched dependency graph can lead to an increased number of possible - perhaps completely different - candidate trees and also generation traces consequently. In the given example case, the new $TcUnit_6$ enables an absolutely different $Trace_2$ for **A** while relying on already existing *TcUnits* of the ecosystem:

$$\begin{aligned} TcUnit_6 &: A \Leftarrow D \\ Trace_2(A) &: TcUnit_5 \rightarrow TcUnit_4 \rightarrow TcUnit_6 \end{aligned} \tag{4}$$

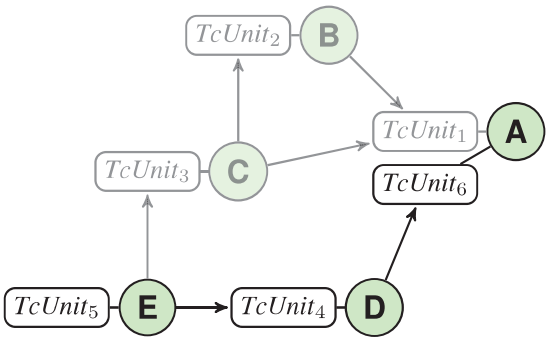

Figure 2: Extended dependency graph of *TcUnits* which enables a new generation trace.

Both, $TcUnit_1$ and $TcUnit_6$, are eligible for generating **A**. Therefore, the respective connections between **A** and the two *TcUnits* shown in **Figure 2** express alternatives to choose from, whereas dependency arrows are always strictly required. Another possible extension would be a *TcUnit* which allows for generating a completely new data type while maybe relying on existing *TcUnits*. Owing to the modularity of *TcUnits*, it is very easy to extend and exchange parts of the ecosystem. However, the level of extensibility is mainly determined by the concrete use and chosen structure for *TcUnits*.

When designing *TcUnits* for a concrete use case, we propose to divide and classify single contributions into different domain parts, i.e. robot-specific computations, geometrical basics, etc., as well as application domain specific parts of the process (e.g. automotive industry or composite manufacturing). This way, specific parts or behaviours of the generated process can be systematically influenced or modified by replacing or adding particular *TcUnits*. That might be the supply of new alternative actuators, the replacement of an improved algorithm, or an extension which leads to tasks being able to be performed in a new different way (by different traces). The latter implies, that when adding a new *TcUnit* for solving a particular problem by relying on already computable results, the generation capacity of the ecosystem is extended whereat functionality of already existing *TcUnits* can be reused. Even *TcUnits* of equivalent result types can, however, contribute to make one problem solvable by using different actuators, resources or techniques.

In particular, applications solving similar low batch size production or consisting of a huge number of similar tasks can profit from this result-finding architecture. Which *TcUnits* are needed to generate a desired result, as well as their execution order, highly depends on amount and nature of all available *TcUnits* in the ecosystem. The navigation through the generation order - where some *TcUnits* compute their results silently and others might request expert input over UI - can be arranged highly interactive and tailored to the particular use case.

2.3 Expert involvement

As a main goal, we addressed the involvement of domain experts in our proposed architecture. While programmer

and mathematical experts identify, design and implement *TcUnits* needed for a given application domain, experts of the overall domain themselves should be involved at the latest when it comes to generate concrete application processes. This is conceptually included in two different levels in the architecture:

1. When generating a specific type, usually multiple generation traces exist: some just having permuted orders of *TcUnits*, but also some basing on different *TcUnits*. Especially for the latter case, the choice of the one trace to use can be delegated to the expert, who knows about even minimal differences of impact on the result quality, for example, depending on the concrete *TcUnits*.

2. At some point, process definitions often need information provided by a domain expert in regard to concrete, present facts (which e.g. evolve from a foregoing process). The presented architecture intents *TcUnits* being able to rely on expert input when generating. In a concrete implementation of the architecture, we propose to give *TcUnits* the possibility to initiate and handle UI-actions.

In addition, *TcUnits* requiring expert input might compute example values which can be provided as suggestions to the expert. This helps to make the expert focus on solving problems which can not be computed automatically. In some use cases, the generation for the same result is repeated iteratively, that means an existing generation trace is executed again, in order to modify and improve the result by entering slightly adjusted expert inputs. When intermediate user interaction is requested, the generation mechanism can set up input values from a previous generation, which makes it easier for the expert to adjust previously entered values.

3 Application to CFRP production

The concept of *TcUnits* has been used as base architecture in an off-line programming platform for defining processes in the domain of carbon-fibre-reinforced polymers [3]. This off-line programming platform was part of a project together with the German Aerospace Center, Center for Lightweight Production Technology (DLR-ZLP), and aimed at semi-automatically specifying robot-based manufacturing processes for CFRP plane parts [7]. Thereby, hundreds of differently shaped CFRP textiles (cutpieces) of up to $2m^2$ need to be laid out (i.e. draped) in a concave form in multiple layers in order to build one CFRP component. High pickup and draping precision and the preforming of cutpieces to their three-dimensional shapes play an important role for the overall manufacturing quality, which is, of course, extremely important in the domain of aerospace. For the preforming, three diverse particularly designed grippers using different preforming strategies have been built and should be evaluated in respect to draping quality and accuracy.

All the different related domains of this application, i.e. aerospace, CAD and construction, modular and exchangeable hardware, CFRP, etc., are a perfect match for an integration with our presented architecture based on *TcUnits*. For this, the process of draping a CFRP textile has to be analysed, and single contributions of different domains have to be identified which can be solved by concrete *TcUnits*.

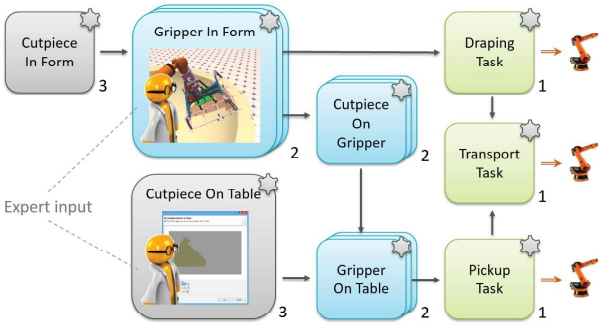

Figure 3: *TcUnits* used in CFRP production example. Their result types are not shown explicitly but are indicated by the *TcUnit's* name.

The generation of process definitions is performed "backward-oriented" [3]. That means, the position of the cutpiece on the gripper is not determined by the pickup but by the later draping within the form. However, since the pickup as well as the draping movement both depend on one cutpiece position on the gripper, the backward-orineted approach thus facilitates the important parts of the process which are primarily responsible for high precision and quality. **Figure 3** illustrates the concrete use of *TcUnits* in the offline programming platform which allows for a user-guided, semi-automated definition of process descriptions for handling multiple cutpieces. Every numbered box stands for a *TcUnit*, connected by arrows which denote their dependencies. Their result types are implicitly indicated by the names of the respective *TcUnits*.
Each *TcUnit* numbered with **1** (green) is responsible for generating concrete tasks for robots. The generation of *TransportTask*, for example, results in a transfer movement between the final position after *PickupTask* and the start position before *DrapingTask*, which are thus required inputs for *TransportTask*. Those numbered with **2** (blue) are domain specific *TcUnits* with respect to the chosen gripper and have been provided for a variety of different grippers each. Depending on the concrete gripper, the respective *TcUnit* is chosen for generation traces. Amongst others, this includes the optimal gripper position when e.g. draping the textile into the concave form depending on the particular gripper geometry. *TcUnits* numbered with **3** (gray) contain data-supply (CAD) and computations within the CFRP domain.
TcUnit CutpieceInForm, for example, retrieves its result by loading the geometrical target position of the cutpiece from the engineer's construction plan. Having this information, an optimal *GripperInForm* position can be specified which optimally fits the transformed gripper onto the

target position of the cutpiece. Herewith, robot and gripper movements can be generated to perform the *Draping Task*. Additionally, the *CutpieceOnGripper* position is implicitly given and can be computed. Along with the *CutpieceOnTable* position, the position of the gripper in order to pick up the cutpiece can be calculated. Now, the concrete robots' *PickupTask* and subsequently the *Transport Task* can be obtained.

3.1 Expert involvement

Most *TcUnits* of Figure 3 are either data providers or generate their results by geometric and actuator-specific computations. However, *GripperInForm* and *CutpieceOnTable* require domain expert knowledge and use interactive UI requests when triggered to contribute their results. In the offline programming platform, two kinds of user-interactive dialogs are provided for the expert to find optimal results for the two *TcUnits*.

Figure 4: *TcUnit* acquiring expert input for ideal gripper position when draping.

Figure 4 shows a dialog initiated by *TcUnit GripperInForm*: the robot can be moved and the transformation of the gripper can be set by the expert. A visualization always shows the robot cell configuration as it would be like when the draping process would be performed with the current inputs. It thus allows the user for interactively experiencing with different possibilities. Within the dialog, the coverage of vacuum modules of the gripper above the cutpiece is indicated for the current input values and already gives feedback about quality and if the cutpiece can adhere on the gripper et all. Furthermore, the collision distances between vacuum modules and the form are calculated by a physics engine and are indicated coloured in the dialog in order to support the expert when positioning the gripper.

A second dialog (see **Figure 5**) provides the draggable and rotatable contour of the cutpiece on top of the supplier table and aims for specifying the original *CutpieceOnTable* position from where the cutpiece can be picked up by the gripper. Snap-ins, zoom, coordinate-extraction and table-border detection support the expert to specify the cutpiece position in sub-millimetres accuracy.

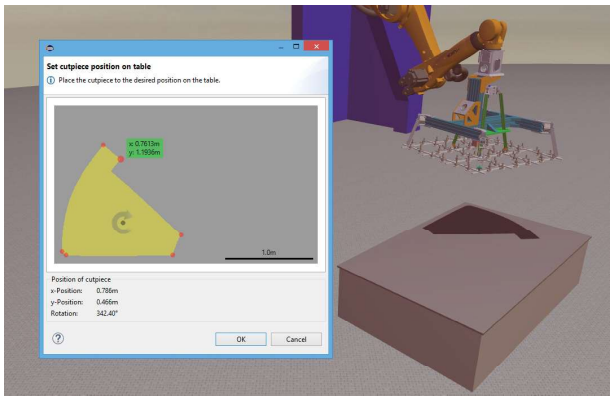

Figure 5: *TcUnit* acquiring expert input for cutpiece position on supplier table.

3.2 Extensibility

By using the presented architecture of *TcUnits*, it was very easy to add functionality for multiple grippers. For each gripper, specific *TcUnits GripperInForm*, *CutpieceOnGripper* and *CutpieceOnTable* had to be provided. In order to support a new, fourth gripper to perform the manufacturing process, new implementations of only these three *TcUnits* would have to be provided accordingly.

Owing to the complexity of some grippers, some *TcUnits* show up UI-elements and rely on expert input. For *TcUnit GripperInForm* (see **figure 4**), an alternative *TcUnit* had been implemented which uses an evolutionary optimization algorithm instead. This algorithm retrieves genetic variations of possible draping positions of the gripper along with their collision information from the physics engine. The results are evaluated relating to their respective fitnesses, i.e. the coverage of vacuum modules on the textile and their collision distances to the form. Thus, the platform had been extended by an alternative (more automatized) way how to generate draping processes for textiles.

For an additional level of abstraction, the offline programming platform had conceptually been designed as a service-oriented architecture (SOA) [8]. In the java-based implementation, all *TcUnits* of the platform have been integrated as OSGI services [9] which implement the interface of *TcUnits*. The generation logic now dynamically retrieves *TcUnits* by using the standard OSGI mechanism. This allows the platform for dynamically activating and deactivating specific *TcUnits* during runtime. This way, the expert can define which *TcUnits* are enabled for the next generation. For example, the expert based *TcUnit GripperInForm* can be disabled while the new one basing on the genetic algorithm is enabled.

3.3 Experimental results

In the offline programming platform as a case study for the concept of *TcUnits*, the resulting robot tasks (*PickupTask*, *TransportTask* and *DrapingTask*) are both, convertible to *Kuka Robot Language* (KRL) and compatible

to the robotics framework *Robotics API* [10]. Whereas KRL had been used to bring the programs onto real robot controllers, the Robtoics API moreover enabled a simulation of the manufacturing process where robots and grippers perform the cutpiece handling. Thus, all three grippers could be successfully evaluated each with different cutpieces in simulation and real-world runs. Owing to the modularity of the presented architecture of *TcUnits*, it was easily possible to also exchange the robot cell in which the manufacturing process is performed. Hence, the process could be defined and executed in either the "Multi Functional Cell" (MFC) [11] or the "Technology Evaluation Cell" (TEC). In **Figure 6**, the manufacturing process with the same end-effector, i.e. the so-called Grid Gripper [7], is shown in both robot cells.

The manufacturing of three different CFRP textiles of up to 2 square metres have been fully evaluated: defining the process by the presented concept of *TcUnits*, converting the process definitions to runnable code, preform and drape the real cutpieces into the form. The evaluation results of the platform basing on *TcUnits* have been very good: the programming effort could drastically be reduced compared to conventional robot teaching. Whereas manual robot teaching took about half an hour up to one hour per textile, the defining of the process by an expert with our platform only took about 2 up to 7 minutes per textile, while draping accuracy of textiles still could be performed within sub-millimetres. More detailed results in respect to quality, accuracy and time-efficiency are presented in [3].

Figure 6: Evaluating the manufactruing process with the Grid Gripper in both the MFC (top) and the TEC (bottom).

4 Related work

An overview of current robot offline programming tools can be found in [12]. As mentioned before in [3], presented solutions are powerful general purpose tools (e.g. *WorkVisual* [13], *Robotics Suite* [14] or *Easy-Rob* [15]), where the process is usually specified with basic robot movements (e.g. movements along linear, circular or spline paths). But most of these tools do not support domain specific, semi-automatic program generation on the abstraction level of process steps. There are also tools specialized for concrete domains (e.g welding) or even for a different type of CFRP production called Automated Tape Laying (e.g. Vericut [16], Microplace [17]). Those tools usually do not provide a process-oriented guidance for more complex manufacturing tasks or interactive planning support. In contrast, we presented a general concept for a software architecture of programming platforms that enables extensibility to a new domains without extensive programming effort which is not the case regarding current tools.

Robot programming by demonstration (*PbD*) [18] is another, powerful, online method besides offline programming with many different approaches (e.g. [19, 20, 21]). For example, with *KINETIQ Teaching* for *Yakasawa* robots, a robot operator can teach welding trajectories with hand guiding the welding tip to desired positions [22]. Generally, *PbD* reduces required programming knowledge due to minimizing explicit programming of robot tasks. But for production processes with a large number of production steps, process definition with *PbD* without any automatic program generation or guidance would be very time consuming. Traditional PbD systems use a teach-pendant to demonstrate the desired robot motion, but teaching large robot systems with heavy tools can still be problematic. In this case, offline programming tools have advantages that they can display the scene in the robotics cell from different perspectives or visualise different parameters that are not necessary obvious during the teaching. *PbD* could be very well integrated in the presented architecture in form of a *TcUnit* with user input.

For handling complex robot task programming in uncertain environments, there are several concepts of sensor-based programming as proposed e.g. in [23, 24, 25, 26]. Another example is the sensitive lightweight robot arm *LBR iiwa* from KUKA. It has integrated joint torque sensors that make contact detection possible. Therefore it can be used for tasks, where the position of the object can be determined sensitively. In most cases, these approaches have preplanned or offline calculated trajectories and they try to handle differences in geometric models and reality by position control with different sensors. Again, they minimize the amount of explicit programming, but they lack of process-oriented guidance or interactive planning support. On the other hand, offline programming requires detailed geometric models that are usually available for industrial robot cells and production parts, so that the production process can be computed au-

tomatically.
One of the most common formalism for describing planning domains is PDDL [27]. In PDDL, there is a domain description (based on predicates) and descriptions of possible actions with preconditions and their effects. The desired goal state and initial conditions are part of the problem description. The planning environment tries find a sequence of actions that lead to the desired goal state. In presented architecture, the planning is done by combining *TcUnit*s with suitable inputs and outputs to get a generation trace. Therefore, the set of available *TcUnit*s in the ecosystem and their compatibility, play the major role for planning. If various generation traces have been found, one has to be chosen. This can be done manually by domain experts or automatically, e.g. based on preferences or some heuristic.

5 Conclusion

In this paper, we presented a software architecture for semi-automatically programming manufacturing processes with industrial robots. Therefore, we introduced the concept of *TcUnits* which provides a flexible and dynamical ecosystem of modules. Each module can solve different task, e.g. the planning of robot motions, the computation of end-effector poses, or the query of expert user input. Especially, the involvement of expert knowledge and its integration into robot programming is a key contribution which improves the current situation. When a manufacturing process is modified, the ecosystem of *TcUnits* can be extended with new modules and searched for new solutions.
We have implemented the proposed software architecture into an off-line programming platform tailored to the needs of the CFRP manufacturing domain. For evaluation purposes, we have been generating robot programs for two different robot cells with three different handling end-effectors [7, 3]. The evaluation successfully confirmed that the concept of *TcUnits* performs very well and increases the productivity of defining robot-assisted production processes. Compared to conventional manual teaching, which is still state of the art, we could accelerate robot programming tremendously, i.e. we could reduce the time from 66 to 7 minutes for one exemplary cut-piece. This clearly shows that the proposed software architecture can contribute to the evolution of industrial automation towards an *Industry 4.0*.

References

[1] International Federation of Robotics. Industrial robot statistics. [Online]. Available: http://www.ifr.org/industrial-robots/statistics/

[2] J. N. Pires, "New challenges for industrial robotic cell programming," *Industrial Robot*, vol. 36, no. 1, 2009.

[3] L. Nägele, M. Macho, A. Angerer, A. Hoffmann, M. Vistein, M. Schönheits, and W. Reif, "A backward-oriented approach for offline programming of complex manufacturing tasks," in *2015 The 6th International Conference on Automation, Robotics and Applications (ICARA 2015)*, Queenstown, New Zealand, 2015.

[4] A. Angerer, M. Vistein, A. Hoffmann, W. Reif, F. Krebs, and M. Schönheits, "Towards multifunctional robot-based automation systems," in *Proc. 12th Intl. Conf. on Inform. in Control, Autom. & Robot., Rome, Italy*, 2015.

[5] J. Craig, *Introduction to Robotics: Mechanics and Control*, 3rd ed. Prentice Hall, 2005.

[6] M. Hägele, K. Nilsson, and J. N. Pires, "Industrial robotics," in *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Berlin, Heidelberg: Springer, 2008, ch. 42, pp. 963–986.

[7] T. Gerngroß and D. Nieberl, "Automated manufacturing of large, three-dimensional CFRP parts from dry textiles," in *SAMPE EUROPE Technical Conf. & Table-Top Exhib.*, Sep. 2014.

[8] T. Erl, *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. Prentice Hall PTR, 2005.

[9] OSGi Alliance, "OSGi Core Specification, Release 6," URL: https://osgi.org/download/r6/osgi.core-6.0.0.pdf, 2014.

[10] A. Angerer, A. Hoffmann, A. Schierl, M. Vistein, and W. Reif, "Robotics API: Object-Oriented Software Development for Industrial Robots," *J. of Software Engineering for Robotics*, vol. 4, no. 1, pp. 1–22, 2013. [Online]. Available: http://joser.unibg.it/index.php?journal=joser&page=article&op=view&path=53

[11] F. Krebs, L. Larsen, G. Braun, and W. Dudenhausen, "Design of a multifunctional cell for aerospace CFRP production," in *Advances in Sustainable and Competitive Manufacturing Systems*, ser. LNME. Springer, 2013, pp. 515–524.

[12] Y. Gan, X. Dai, and D. Li, "Off-line programming techniques for multirobot cooperation system," in *International Journal of Advanced Robotic Systems*. InTech Europe, 2013.

[13] KUKA Robotics. KUKA.WorkVisual. [Online]. Available: http://www.kuka-robotics.com/en/products/software/engineering_environment

[14] Stäubli. Robotics Suite. [Online]. Available: http://www.staubli.com/en/robotics/robot-software/pc-robot-programming-srs/

[15] EASY-ROB. EASY-ROB 3D Robot Simulation Tool. [Online]. Available: www.easy-rob.com

[16] CGTech. Vericut composite programming. [Online]. Available: http://www.cgtech.de/products/composite-applications/vcp/

[17] Mikrosam. Mikroplace. [Online]. Available: http://www.mikrosam.com/new/article/de/advanced-off-line-composite-programming-software/

[18] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, "Robot programming by demonstration," in *Springer handbook of robotics*. Springer, 2008, pp. 1371–1394.

[19] D. R. Myers, M. J. Pritchard, and M. D. Brown, "Automated programming of an industrial robot through teach-by showing," in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 4. IEEE, 2001, pp. 4078–4083.

[20] R. Zöllner, O. Rogalla, R. Dillmann, and M. Zöllner, "Understanding users intention: programming fine manipulation tasks by demonstration," in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, vol. 2. IEEE, 2002, pp. 1114–1119.

[21] Y. Yokokohji, Y. Kitaoka, and T. Yoshikawa, "Motion capture from demonstrator's viewpoint and its application to robot teaching," *J. Robot. Syst.*, vol. 22, no. 2, pp. 87–97, Feb. 2005. [Online]. Available: http://dx.doi.org/10.1002/rob.v22:2

[22] Robotiq. Kinetiq Teaching. [Online]. Available: http://robotiq.com/products/robotic-welder/

[23] M. H. Raibert and J. J. Craig, "Hybrid position/force control of manipulators," *Journal of Dynamic Systems, Measurement, and Control*, vol. 103, no. 2, pp. 126–133, 1981.

[24] L. E. Weiss, A. C. Sanderson, and C. P. Neuman, "Dynamic sensor-based control of robots with visual feedback," *Robotics and Automation, IEEE Journal of*, vol. 3, no. 5, pp. 404–417, 1987.

[25] J. De Schutter, T. De Laet, J. Rutgeerts, W. Decré, R. Smits, E. Aertbeliën, K. Claes, and H. Bruyninckx, "Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty," *The International Journal of Robotics Research*, vol. 26, no. 5, pp. 433–455, 2007.

[26] B. Finkemeyer, T. Kröger, and F. M. Wahl, "Executing Assembly Tasks Specified by Manipulation Primitive Nets," *Advanced Robotics*, vol. 19, no. 5, pp. 591–611, 2005.

[27] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL - The planning domain definition language," 1998.