Developing Self-Organizing Robotic Cells using Organic Computing Principles

Alwin Hoffmann, Florian Nafz, Andreas Schierl, Hella Seebach, and Wolfgang Reif

***Abstract*— Nowadays industrial robotics applications, which are often designed and planned with a huge amount of effort, have a fixed behavior during runtime and cannot react to changes in their environment. Failures can hardly be compensated and often can only be repaired by human involvement. In this paper we present a layered approach for developing self-organization systems that are able to take advantage of Organic Computing principles and therefore are more robust and flexible during runtime.**

I. Introduction

With respect to their structure, traditional automation systems are very static. The material flow is fixed and every component is optimized according to the planned system structure to reach maximum throughput. This approach is very suitable for mass production as in the automotive industries, where one product is manufactured for a considerable time. Even the use of industrial robots does not change this situation. In fact, industrial robots are very flexible and, given an appropriate tool, are able to perform a large variety of tasks. However, the complex and tedious programming of today's industrial robots, the fixed wiring and difficult integration of additional devices, as well as the very static layout of shop floors do not exploit the possible flexibility of robotic solutions. As a consequence, high effort is needed to customize and adjust automation systems, making them hardly applicable for small-series production with flexible products where regular adaptation is required. Moreover, failure tolerance and flexible optimization is hard to achieve with traditional automation systems.

On the other hand, the idea of Organic Computing [1][2] and Autonomic Computing [3] is to develop systems which possess self-x properties, like self-healing (compensation of failures), self-adaptation (adaptation to changing or new jobs and system structure) or self-optimization (optimization according to a given fitness function). In the context of production automation, the goal is to provide architectures and techniques to build *organic* automation systems, where organic means life-like behavior or more concise that the systems are capable of autonomously adapting to changes in their environment. This behavior is often realized by the use of bio-inspired paradigms and algorithms, e.g. genetic algorithms or pheromone-based approaches.

Hence, the well-designed combination of Organic Computing principles and robot technology can lead to hyper flexible and robust automation systems. While robots, mobile platforms as well as industrial manipulators provide mechanical flexibility and the ability to perform a large variety of different tasks, Organic Computing introduces self-organization to the systems which enables them for self-healing, self-adaptation or self-optimization. For example, an organic automation system can compensate failures due to its self-healing capabilities and continues to operate in *graceful degradation* – a requirement for robotic systems which is getting more and more important [4]. Especially in small and medium enterprises, where there are phantom shifts at night, systems of this kind are welcome. Due to self-organization, organic systems are reconfigurable by design and are easily able to adapt to new tasks or products – a requirement in today's globalized economy with its turbulent markets and fast changing demands [5]. However, evolutionary approaches and bio-inspired principle which solely rely on the idea of emergence cannot directly be applied to production systems. Emergence rather has to be controlled and directed to accomplish a defined goal, i.e. manufacturing a product.

In this paper, we want to outline how self-x properties and Organic Computing principles can be applied to robotic cells. Sect. II describes the challenges that are to facing in order to build organic robotic cells. The challenges are ranging from uncontrolled emergence over the layout of robotic cells to limitation in software architectures. In Sect. III, we present a multi-layer architecture to design and implement organic robotic cells in automation. Based on that architecture, we illustrate our approach with a case study in Sect IV. Finally, conclusions are given in Sect. V.

II. Challenges

From our point of view, the development of self-organizing robotic cells using Organic Computing principles poses three major challenges:

1) To render organic systems acceptable for industry, *emergence must be controlled* to accomplish a defined goal.
2) To apply self-x properties, the layout of robotic cells must provide *additional degrees of freedom*.
3) To utilize these additional degrees of freedom, robotic software architectures must provide *flexibility with regard to programming techniques*, coping with geometric uncertainty and device integration.

These challenges and their influence on the development of organic automation systems are described in detail in the following sections.

Alwin Hoffmann, Florian Nafz, Hella Seebach, Andreas Schierl, and Wolfgang Reif are with the Institute for Software and Systems Engineering, University of Augsburg, D-86135 Augsburg, Germany.

This work has been partly sponsored by the priority program *Organic Computing* (SPP OC 1183) of the German research foundation (DFG).

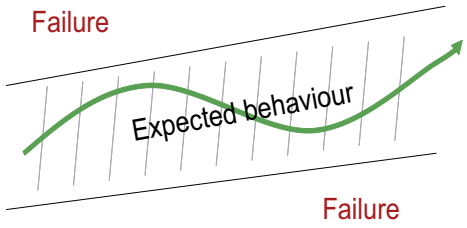

Fig. 1. Organic production systems require a corridor of expected behavior. Inside this corridor, emergent behavior is approved and even desired.

A. Controlling Emergence

A main concept of self-organizing systems is emergence. Emergence describes the appearance of complex system behavior caused by relatively simple and local interactions of individuals without the control of a central instance. Hence, the system behavior is not explicitly programmed, but a result of these local interactions. An example of emergent behavior is an ant colony where no central control is present. Instead, each ant is an autonomous unit that reacts depending on local information, i.e. pheromones, and genetically encoded rules.

Thus, the behavior of the individual components cannot be exactly predicted. Müller-Schloer [6] calls this kind of behavior *bottom-up constraint propagation* which stands in contrast to the classical top-down design of technical systems. In the latter approach, the developer tries to model and implement all possible system states. This usually starts with a high-level specification, until after a number of transformations and refinements, executable code is generated.

However, having an exhaustive model of a complex system is often not feasible and even contradictory to the idea of emergence. In order to solve this contradiction, we suggest defining a corridor of *good* expected behavior [7] for every organic production system. Inside this corridor, emergent behavior is approved and even desired, whereas the system is in an exceptional state when this corridor is left (cf. Fig. 1).

This corridor is defined through constraints by the system developer and allows *controlled emergence*. Usually, these constraints can be observed locally by each autonomous component of the system. If one or more constraints are violated, the component tries to restore the constraints locally. If this is not sufficient, it starts to involve surrounding components until a valid solution is found that satisfies the constraints. In Organic Computing this kind of architecture is called an Observer/Controller architecture [6] [8]. By doing so, organic automation systems are self-organizing and can be directed to accomplish a defined goal, e.g. to manufacture products. Besides, behavioral guarantees in terms of functional correctness can be given [9].

B. Adding Degrees of Freedom

Usually, automation systems are designed and tuned to accomplish pre-defined tasks for a long period. In single-station automated cells, a production machine is typically equipped with a material handling system (e.g. a robot for loading and unloading the machine) and a storage system. Due to this setting, the cell is able to operate unattended but the system fails if any of the components breaks. An automated production line consists of multiple workstations that are automated and linked by a transport system which transfers parts from one station to the next. Again, if one components breaks, the whole system fails. According to [10], flexible manufacturing systems still have limited capabilities with regard to customized products and failure compensation. They even state that the need for flexible products and adaptive systems cannot be supplied with traditional approaches.

In order to become self-organizing, automation systems require additional degrees of freedom and redundancy in the available hardware. Without these prerequisites, the system is not able to adapt to new environmental conditions or to compensate failures:

- For *self-healing*, an organic automation system needs redundant hardware components. Otherwise, it cannot compensate for the failure of one component and continue operation in graceful degradation.
- Regarding *self-adaptation*, an organic production system needs degrees of freedom, i.e. flexible tools or transport systems, in order to adapt to changing or new tasks as well as to a modified system structure.
- Finally, *self-optimization* is only possible if there are several degrees of freedom which can be optimized with respect to a given fitness function.

Due to these reasons, we believe that robotic cells are well-suited for self-organization by using Organic Computing. In robotic-based systems, additional degrees of freedom can be achieved by adding robots, redundant tools, or tool-changing systems. Concerning transportation, robots can be connected using carousels, two-way conveyors, or even mobile platforms. Further details are given in Sect. III, but here it is worth mentioning that the concrete choice of how redundancy is added can impact the system's robustness and its mean time to failure, as the example in Sect. IV shows.

C. Requiring Software Flexibility

By adding degrees of freedom and redundancy to the available devices and to the shop floor layout, self-organization becomes feasible. However, to completely utilize self-x properties, additional requirements to the architectures of robotic systems with regard to software flexibility are necessary.

Flexible and reconfigurable automation systems require the introduction of *smart products* carrying information about how to be processed by the system. This can be e.g. realized by using RFID [10]. As a consequence, a *product-centric* approach of configuring and commanding industrial robots and their tools is required. Pre-defined motion sequences have to be replaced by more dynamic motion planning considering the environment and avoiding obstacles. Due to the dynamic system behavior, the use of previously taught motions cannot be sufficient anymore. Instead, the use of sensor feedback (e.g. vision) or compliant devices should be considered. With sensor-based or compliant motions [11], an error-tolerant execution of complex robot tasks in uncertain and unknown environments is possible [12].

In contrast to these demanding requirements, industrial robots are still programmed with special robot programming languages which are derived from early imperative languages and have not evolved much since then. Due to these low-level programming techniques, developing software for an industrial robot is a complex and tedious task requiring considerable technical expertise [13]. Hence, industrial robots are usually equipped and programmed to perform only a set of pre-defined tasks. This contradiction between low-level programming and high-demanding requirements must be solved in the future to realized self-organizing robotic systems.

Furthermore, the integration of external devices must be facilitated. Today, tools are usually connected by a fixed wiring to a robot controller and communicates over digital and analog I/O ports. But when using tool changing systems, no human interaction should be required. The software of the robot controller must be able to independently cope with different tools mounted to the robot. Moreover, it must be possible to integrate arbitrary sensors for intelligent perception and sophisticated tools that allow e.g. complex grasping strategies and dexterous manipulation [14]. The introduction of plug-and-play mechanisms as proposed in [4] would cover this requirement for flexible device integration.

III. Architecture

Our approach uses a layered software architecture which addresses the system at different levels of abstraction. The proposed architecture is depicted in Fig. 2.

On top of the hardware layer, two software layers are located for controlling the robot. The lower one, the *Robot Control Layer*, is responsible for the real-time critical, low-level hardware control, whereas the upper layer, the *Robot Programming Layer*, is used for defining the control flow and specifying required motions and tools actions. For traditional production systems, these three layers are sufficient, as they allow the robot to execute arbitrary, pre-programmed tasks in a reliable, repeatable fashion. However, to extend the system towards self-organization, additional communication and control software is required.

Therefore, our approach adds two more layers on top, which control the robotic system according to Organic Computing principles. The first layer, the *Organic Control Layer*, wraps the components of the robotic cell and turns them into software agents coordinating with other agents through communication. Furthermore, it is responsible for the execution of *capabilities* which are to be applied to the workpiece. When this layer detects a locally unrecoverable error, the *Organic Planning Layer* takes control and searches for a new configuration to achieve the task. Once a solution has been found, control is returned to the Organic Control Layer for further execution. The layers of the architecture are explained below from bottom up.

A. Hardware

The foundation of each robotic cell is a set of robots with tools that are interlinked with a transportation system.

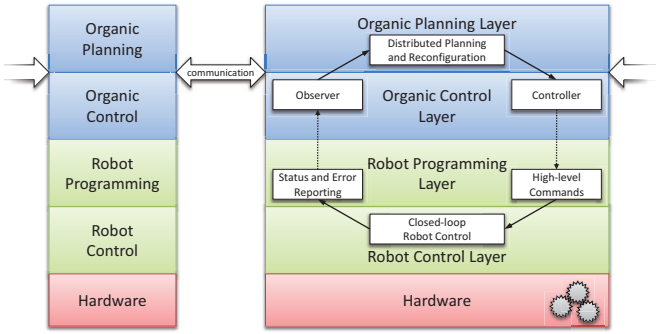

Fig. 2. The proposed architecture for self-organizing robotic cells showing two individual components.

In order to become a self-organizing production system, additional degrees of freedom are required as state in Sect. II-B. This means that a robot cannot only be equipped with one static tool corresponding to its pre-assigned task. For simple cases, it might e. g. be enough to equip the robot with a set of equal drills so that it can replace them when they fail during production, but for exploiting all advantages of self-organizing systems, different tools are needed that can perform a variety of diverse tasks, and a way to interchange them without human interaction. This can be achieved by the use of external tools, by an automatic tool exchange system, or by using advanced tools like anthropomorphic hands which allow dexterous manipulation.

If different tasks have to be executed, or the different task steps should be assigned to different robots, the transportation system also has to become flexible. Instead of a single conveyor connecting the robots in a given order, this set-up requires a way to change the order a workpiece passes the different robots. Similar to existing systems, robots can be connected using a carousel or two conveyors, one moving forward and one moving backwards. Thus, each robot can forward the workpiece to any other robot by placing it onto the right conveyor. Corresponding to the idea of hyper flexible manufacturing systems [4], another solution is to replace the conveyor by a set of mobile platforms navigating between the robots, transporting partly processed workpieces.

Such a system can show a dynamic behavior and, moreover, as the hardware devices are expected to perform different tasks, all of them have to be controlled by a computer-based system. This software layer must provide real-time guarantees to reliably control hardware devices.

B. Robot Control Layer

Low-level hardware control is performed in the Robot Control Layer. It is responsible for applying open or closed loop control laws on actuators and sensors in order to make the hardware execute the requested actions. Therefore it has to be implemented in a real-time capable environment, e. g. running on a micro controller for simple actions or under a real-time operating system (such as VxWorks, QNX or real-time extensions for Linux). For commercial KUKA robot systems, this layer – the so-called *kernel system* – is implemented in VxWorks. It can execute motion commands and

send data to attached tools using fieldbus communication. However, it is quite limited with respect to sensor integration or compliant motion – fields where research robot controllers like OROCOS [15] are more advanced.

As a typical robot action consists of more than the application of one single control law with given parameters (e. g. one motion to a point), the robot control layer has to provide an interface allowing to specify multiple control laws or commands that are to be applied sequentially or in parallel. This interface can be used by the programming layer. Furthermore the control layer has to monitor the attached hardware for errors, and report them to the above layer to allow reasonable failure strategies.

C. Robot Programming Layer

The Robot Programming Layer offers an interface which accepts high-level commands to be executed by the robot. It is responsible for transforming them into control laws that can be executed with real-time guarantees in the robot control layer. Furthermore, it transfers the control laws to the robot control layer and monitors execution progress, errors and sensor events. For KUKA robots, this layer is provided by the robot programming language KRL which allows to write robot programs including extended control flow (e.g. conditional statements and loops), motions and tool commands. However, as the self-organizing robot cell – opposed to traditional production cells – does not have a fixed processing or material flow order, it is not possible to write one program for each robot that can be executed repeatedly to perform the unchanging robot task. Each robot needs a set of robot programs (one for each robot capability) that can be started and controlled from a higher architecture layer.

As the dynamic nature of a flexible production system makes it hard to guarantee exact positioning of the work-pieces during transportation, these systems also have to cope with greater uncertainty about object locations. Thus the integration of sensor feedback for object localization becomes more important here, as well as the possibility to program tolerant or compliant manipulators or tools. Also dynamic motion planning with obstacle avoidance for both robots and mobile platforms must be possible using this layer.

When trying to control a flexible production cell through a set of individual robot programs (one for each robot capability), these programs as well have to be flexible and highly configurable as described in Sect. II-C. However, passing detailed environment information to traditional robot programs is often quite complex, involving fieldbus communication, thus limiting the range of possibilities. These problems can be solved by using a robot control architecture that allows programming robots in standard, high-level programming languages, such as the one described in [16]. It provides a high level, object-oriented API for programming robots which can be directly used from the higher layers or encapsulated into a service that can be e.g. accessed via standard service-oriented methods.

D. Organic Control Layer

The organic intelligence of the system is encapsulated in the top two layers. The main task of these layers is to maintain the behavioral corridor of the system (see Sect. II-A). The specification of this corridor is done by OCL constraints, which are annotated to the particular models during the design process. This design process provides a guideline for development of the organic computing layers of such a system and is described in detail in [8]. These OCL constraints define an invariant over the system state and distinguish good from erroneous states. More informal, they describe how correct configurations of the robots must look like. The organic intelligence then tries to preserve these invariants as long as possible and in case of a violation to restore them, by reconfiguring the system. For a detailed description of this restore invariant approach see [7]. One major benefit is that the behavior of the system components, i.e. the robots, is formalized and verified, so we can – in combination with the restore invariant approach – guarantee that the system stays in the defined corridor.

The Organic Control Layer consists of two main components. The *Observer* evaluates the invariant based on the status information it receives from the lower layers. Therefore, an interface which allows to receive feedback from the Robot Programming Layer (e.g. example error-messages or sensor data) is needed. Whenever the observer detects a violation of the invariant, it activates the planning layer and forwards all gathered information. The second component in this layer is the *Controller*. It performs the capabilities assigned by the planning layer and commands robot actions required to apply the capabilities and exchange resources. It makes use of the interface provided by the Robot Programming Layer and controls the robot to ensure that the right capability is applied. It further reacts to new configurations sent by the Organic Planning Layer, for instance to change the performed actions of the robot.

E. Organic Planning Layer

The Organic Planning Layer is responsible for calculating new configurations if an error occurs and the invariant is violated. It analyzes the current situation and, as most of the failures cannot be compensated by a robot alone, it has to communicate with the planning layers of other robots to gather information about available robots and their capabilities. Then, the planning component tries to find a common solution to reach the objectives. After a consensus is found, the planning layer forwards the new configuration for its responsible robots to the Organic Control Layer, which then commands the robot accordingly.

The advantage of moving all the self-organization into this high-level layer is to use the full bandwidth of planning approaches, like bio-inspired or genetic algorithms as well as simple planners. This layer provides a kind of plug-in interface to allow the use of several methods and algorithms for coordination and planning, implemented as centralized or decentralized variants. System architects can choose what is best suited for their kind of system and problem to solve.

IV. Example

In this section we want to illustrate the presented approach on a vision of a future adaptive production cell. It shows the benefits of applying organic principles to traditional robot systems. Traditional engineering would handle and design such a production cell in a rather static way, consisting of individual machines that process workpieces with their tools and linked to each other in a strict sequential order using conveyors or similar mechanisms. The layout of the cell is therefore predefined, very inflexible, and rigid. Additionally, and maybe more important, such a system is extremely prone to system errors as the failure of one component will stop the whole system. However, the adaptive production cell is self-organizing which means that it is adaptive according to user-defined tasks (work plans) and compensates for component failures. Furthermore, it tries to optimize the throughput by finding a configuration which is best suited for the actual work plan.

The adaptive production cell consists of KUKA Lightweight Robots (LWR), which are capable of using different tools. The traditional conveyor belt has been replaced by flexible and autonomous transportation units, which can carry workpieces. The goal of the cell is to process workpieces in a user-defined sequence of tool applications (work plan).

Sect. II expounds that redundancy and soft flexibility are needed to enhance traditional systems with self-organization. To achieve the maximum benefit from redundancy, it is important how the redundancy is distributed within the system. For example, it would be possible that a robot has the same tool three times and is the only one with this tool. Then, the robot is capable of reacting two times on tool breaks, but the breakdown of the whole robot stays a single point of failure. To find good distribution strategies for the redundancy, the ADCCA[1] technique [17] technique can be used, which calculates minimal combinations of failures which lead to standstills of the whole production system.

According to these results, the case study is arranged as follows: three LWRs for processing, four carts for transportation and two storages, which provide unprocessed workpieces and store finished ones, are used. Each LWR is able to perform all three capabilities: drill a hole into a workpiece, insert a screw into the drilled hole and tighten the inserted screw. The user-defined standard work plan is to process all workpieces with all three tools. The given order is first to drill the hole, then insert the screw and at last to tighten it. In principle, an easy but not very high-performance solution is to let each robot perform all capabilities and change tools after each step. As switching tools is very time consuming compared to the time for applying the capability, the standard configuration is to let every robot perform a different task. The distribution of processing steps among different robots requires flexible routing of carts so that the correct order is maintained. One such configuration is sketched in Fig. 3.

If a failure occurs now, e.g. the drill tool of the drilling robot breaks, the robot monitors this failure and starts a

[1]Adaptive Deductive Cause-Consequence Analysis

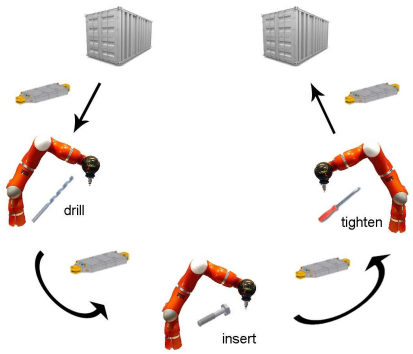

Fig. 3. Adaptive production cell

reconfiguration. It collects the information of the neighboring robots and carts, calculates a new distribution of tool assignments and re-routes the carts in a way, that production can continue. A traditional system would stop and a human interaction would be needed here. The reconfigured situation is depicted in Fig. 4.

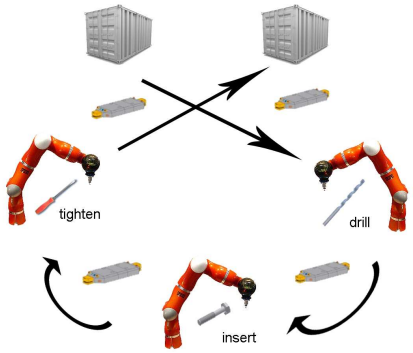

Fig. 4. Adaptive production cell

In this case study, the robots and carts have only local rules and interaction possibilities. The resulting whole system is a self-organizing production cell which is capable of reacting to changes in the environment and new work plans. The configurations which are calculated by the robots in case of local invariant violations (e.g. capability loss) restore the invariants specified for the system which means that the occurring emergence stays in an expected behavioral corridor as claimed in Sect. II-A.

The organic layers are implemented with a multi-agent framework called Jadex [18], which also provides the communication infrastructure. On each robot one Organic Control Layer agent is running and coordinating the robots via the interfaces provided by the Robot Programming Layer. Whenever a failure occurs or a reconfiguration request of another robot is received it spawns an Organic Planning Layer agent which then is handling the reconfiguration for this robot. Currently, reconfiguration is done by solving a constraint satisfaction problem to receive valid configuration for each robot. The constraints originate from the invariants

formulated over the system which describe how correct configurations must look. More details can be found in [9]. For a proof of concept, we implemented the robotic-specific layer using the simulation environment of Microsoft Robotics Developer Studio and placed them under the organic layers. For more information see [19].

V. Conclusion

In the field of Organic Computing, we were looking at the domain of production automation, in particular the field of adaptive production cells. Further in robotics research, we looked at facilitating the software development for industrial robots and improving software quality. In the presented paper we wanted to show how both worlds can fit together and how Organic Computing principles can be used to realize a flexible automation system. To be more concise, how the architecture of a self-organizing robotic cell can look like and how it can be implemented.

In earlier work, the lower layers were prototypically substituted by a simulation and coupled to the implementation of the organic layers within a multi-agent system, as described in Sect. IV. Nevertheless, we have been able to show that these systems can benefit from the application of Organic Computing principles, especially in terms of failure tolerance and flexibility. One major advantage of the proposed architecture and its implementation is that it is formal grounded and, therefore, allows to give behavioral guarantees with respect to the assigned configurations, which always leads to correct processing of the resources. The definition of a behavioral corridor and the assurance of remaining inside this corridor allows flexibility and gives firm guarantees about the system, which is very important for the acceptance in industrial applications. However, the drawback of moving self-organization into high-level layers is that no real-time critical behavior can be considered. Hence, only non real-time critical reconfigurations are possible.

Different production scenarios and factory settings need diverse reconfiguration mechanisms, e.g. completely decentralized coalition formation or wave propagations. We are currently working on different plug-ins for the Organic Planning Layer to enhance it by several reconfiguration algorithm implementations. In order to meet the flexibility requirements as proposed in Sect. II, we are currently extending our robotic software architecture (see [16]) which corresponds to both the Robotic Programming Layer and the Robotic Control Layer. A mid-term goal is to replace the simulation of the adaptive production cell by an implementation using the lightweight robots in our lab.

References

[1] C. Müller-Schloer, C. von der Malsburg, and R. P. Würtz, "Organic computing," *Informatik Spektrum*, vol. 27, no. 4, pp. 332–336, Aug. 2004.

[2] J. Branke, M. Mnif, C. Müller-Schloer, H. Prothmann, U. Richter, F. Rochner, and H. Schmeck, "Organic Computing – Addressing complexity by controlled self-organization," in *Proceedings of the 2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2006)*, Nov. 2006.

[3] J. Kephart and D. Chess, "The vision of autonomic computing," *IEEE Computer*, January 2003.

[4] M. Hägele, T. Skordas, S. Sagert, R. Bischoff, T. Brogårdh, and M. Dresselhaus, "Industrial robot automation," European Robotics Network," White Paper, July 2005.

[5] M. Mehrabi, A. Ulsoy, Y. Koren, and P. Heytler, "Trends and perspectives in flexible and reconfigurable manufacturing systems," *Journal of Intelligent Manufacturing*, vol. 13, no. 2, pp. 135–146, 2002.

[6] C. Müller-Schloer, "Organic computing: on the feasibility of controlled emergence," in *CODES+ISSS '04: Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. New York, NY, USA: ACM, 2004, pp. 2–5.

[7] M. Güdemann, F. Nafz, F. Ortmeier, H. Seebach, and W. Reif, "A specification and construction paradigm for Organic Computing systems," in *Proceedings of the Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, S. Brueckner, P. Robertson, and U. Bellur, Eds. IEEE Computer Society Press (2008), 2008, pp. 233–242.

[8] H. Seebach, F. Ortmeier, and W. Reif, "Design and construction of organic computing systems," in *Proc. 2007 IEEE Congress on Evolutionary Computation*, Singapore, Sept. 25–28 2007, pp. 4215–4221.

[9] F. Nafz, F. Ortmeier, H. Seebach, J.-P. Steghofer, and W. Reif, "A generic software framework for role-based organic computing systems," *Software Engineering for Adaptive and Self-Managing Systems, International Workshop on*, vol. 0, pp. 96–105, 2009.

[10] M. Zaeh and M. Ostgathe, "A multi-agent-supported, product-based production control," in *Proc. 7th IEEE International Conference on Control and Automation*, Christchurch, New Zealand, Dec. 2009, pp. 2376–2383.

[11] M. Mason, "Compliance and force control for computer-controlled manipulators," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 11, no. 6, pp. 418–432, 1981.

[12] U. Thomas, B. Finkemeyer, T. Kröger, and F. M. Wahl, "Error-tolerant execution of complex robot tasks based on skill primitives," in *Proc. 2003 IEEE Intl. Conf. on Robotics and Automation*, Taipei, Taiwan, Sept. 2003, pp. 3069–3075.

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

[14] A. Okamura, N. Smaby, and M. Cutkosky, "An overview of dexterous manipulation," in *Proc. 2000 IEEE International Conference on Robotics and Automation*, San Francisco, USA, Apr. 24–28 2000, pp. 255–262.

[15] R. Smits, T. D. Laet, K. Claes, H. Bruyninckx, and J. D. Schutter, "iTASC: a tool for multi-sensor integration in robot manipulation," in *Proc. IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, 2008.

[16] A. Hoffmann, A. Angerer, F. Ortmeier, M. Vistein, and W. Reif, "Hiding real-time: A new approach for the software development of industrial robots," in *Proc. 2009 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, St. Louis, USA, Oct. 2009.

[17] M. Güdemann, F. Ortmeier, and W. Reif, "Safety and dependability analysis of self-adaptive systems," in *Proceedings of ISoLA 2006*. IEEE CS Press, 2006.

[18] L. Braubach and A. Pokahr, "Jadex BDI Agent System." [Online]. Available: http://vsis-www.informatik.uni-hamburg.de/projects/jadex/

[19] A. Hoffmann, F. Nafz, F. Ortmeier, A. Schierl, and W. Reif, "Prototyping plant control software with microsoft robotics studio," in *Third International Workshop on Software Development and Integration in Robotics. IEEE Intl. Conf. on Robotics and Automation*, Pasadena, USA, May 2008.