# Towards Safe and Secure Organic Computing Applications *

Matthias Güdemann, Florian Nafz, Wolfgang Reif, Hella Seebach

Lehrstuhl für Softwaretechnik und Programmiersprachen
Universität Augsburg
D-86135 Augsburg, Germany
{guedemann,nafz,reif,seebach}@informatik.uni-augsburg.de

**Abstract:** In this paper we present our ongoing work on "Organic Computing". We present an illustrative case study from program automation that uses OC-paradigms to be failure tolerant and to produce effectively. We present a way to build and verify a formal model of a self-adaptive system. We also give further ideas for formal modeling and our ideas of safety analysis of such systems. Another topic is how to build descriptive models and devising development processes for them.

## 1 Introduction

Many of today's systems are designed and implemented in a rather static way. A changing environment and meeting changing requirements brings forth the need for extensive changes in those systems. Critical systems are made failure tolerant by introducing redundancy in many components which in turn often increases space and cost requirements.

A new approach to address these points is to design Organic Computing systems, which have nature inspired paradigms. These include the so called "self-x properties". These properties are used to allow systems to change themselves or their mode of operation if required.

On the other hand, these desirable properties give rise to new challenges, as new methods for modeling, developing, description and verification of these systems have to be devised. We focus on developing descriptive modeling and verification techniques for Organic Computing systems. We are convinced that these are needed to assure acceptance especially in safety critical applications like avionics, automotive and production automation.

An illustrative case study from production automation is presented in Sect. 2, an overview on formal modeling thereof is given in Sect. 3, Sect. 4 gives an overview on current work on safety analysis and Sect. 5 gives an outlook on ideas for descriptive modeling of adaptive systems.

*This research is partly sponsored by the priority program "Organic Computing" (SPP OC 1183) of the German research foundation (DFG)

## 2 Case Study

The case study describes an automated production cell which is self-organizing in case of failures and adaptive to changing goals. It consists of three robots, which are connected with autonomous transportation units.

### 2.1 Description

In the production cell every robot can accomplish three tasks: drilling a hole in a workpiece, inserting a screw into a drilled hole and tightening an inserted screw. These tasks are done with three different tools that can be switched. Every workpiece must be processed by all three tools in the given order (drill, insert, tighten = DIT). Workpieces are transported from and to the robots by autonomous carts. Changing the tool of a robot is assumed to require some time. Therefore the standard configuration of the system is to spread out the three tasks between the three robots, and the carts transfer workpieces accordingly (see Fig. 1).

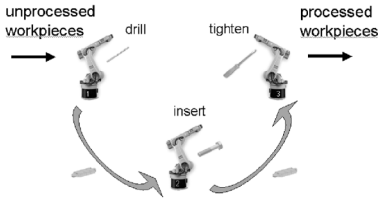


Figure 1: Valid configuration of robot cell

### 2.2 Self-Adaptation

The first interesting new situation occurs when one or more tools break and the current configuration allows no more correct DIT processing of the incoming workpieces. In Fig. 2 the drill of one robot broke and DIT processing is not possible, as no other robot is configured to drill.

However, it is obvious that this is not really a problem, as the robots have three tools and can switch to another tool if one breaks. So it should be possible for the adaptive system to detect this situation and reconfigure itself in such a way that DIT processing is possible again. This implies that at least one other robot also has to switch its tool, so that all three tools are available again.

This can be resolved as shown in Fig. 3. Now the left robot drills, the right robots tightens the screws and the middle robot is left unchanged. For this error resolution, not only the assignment of the tasks to the robots must be changed, but also the routes of the carts and

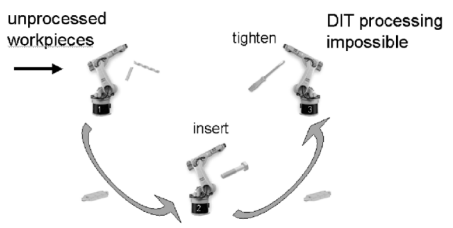


Figure 2: Temporary hazard due to broken drill

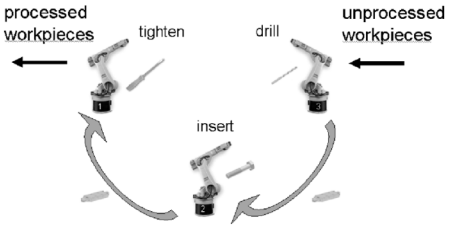


Figure 3: Reconfigured robot cell

the direction of the incoming and outgoing workpieces. If only the tools were switched, the processing of all tasks would be possible, but not in the correct order.

The cell can also exhibit other self-x properties like self-optimization, if additional robots are introduced. Another property would be self-adaption as *graceful degradation*. These are explained in [GOR06].

## 3 Formal Model

Our accomplished and ongoing work focuses mainly on formal modeling of self-adaptive systems. The adaptive principles require special modeling techniques. Nevertheless we were able to do an initial modeling of our example case study using a technique similar to traditional modeling methods. Nevertheless for bigger modeling tasks an approach especially for Organic Computing should be developed.

### 3.1 Restore-Invariant Approach

We see a run of an adaptive system, as exemplified by the adaptive production cell, separated into production phases and reconfiguration phases. We define an invariant that must hold as to allow correct production. Whenever this invariant is violated, a reconfiguration is triggered that restores the invariant as long as this is possible. We call this approach ”restore-invariant” [GOR06].

Its major advantage is that it allows abstraction from an actual reconfiguration algorithm, thus modularizing the modeling and supporting our top-down design approach. Although this may somehow constrict some self-x properties and emergent behaviour [MMB03], it may be required for proving functional properties in safety critical systems [MH95]. The reconfiguration algorithm is just specified via requirements. If a specific reconfiguration algorithm is designed, proving it correct against the specification is sufficient for the correctness of the whole system.

### 3.2 Formalization and Verification of Case Study

We give a short overview of the formalization of the adaptive production cell, a more detailed version can be found in [GOR06]. The system is modeled as transition automata for the Cadence SMV model checker [McM90] using CTL (computational tree logic) and LTL (linear time logic) [EMCJ99].

The cell is the product automaton of the corresponding automata of the robots, the workpieces and the autonomous carts. The robots consist of one transition system, indicating the state of the robot. The workpieces consist of two transition systems, one for the position and the other for the state of the workpiece, i.e. which task has already been completed. The carts are modeled by three automata, one for the position of the cart, one for its configuration (between which robots its tour is) and one for its state (loaded, idle, heading back). The reconfiguration is modeled as control automaton that sends configuration commands to both the robots and the autonomous carts. Additionally, failure automata are defined for every tool of every robot that is in state $yes$ if the corresponding tool fails or in state $no$ if the corresponding tool is usable.

For the model we defined several predicates that appear in the following propositions. The predicates *robotConf* and *cartConf* indicate that the robots, respective the carts are configured, *ditCapable* and *cartCapable* indicate that this configuration is correct, i.e. allows DIT processing and correct transportation of workpieces. To express failures we defined *ditPossible* which indicates that DIT processing is still theoretically possible and *ditFailure* which is true if a robot is configured for a task it can no more perform due to tool failure.

The reconfiguration algorithm is specified using the following two LTL properties that assure that a reconfiguration is correct, as long as a correct reconfiguration is still theoretically possible.

$$\begin{aligned} confDIT &:= \mathbf{G}\left((robotConf \wedge cartConf) \rightarrow \right. \\ &\qquad \left. ditCapable \wedge cartCapable\right) \qquad (1) \\ confCorrect &:= \mathbf{G}\left((Control = EndReconf) \rightarrow \right. \\ &\qquad \left. \mathbf{X}\left(ditPossible \rightarrow \neg ditFailure\right)\right) \qquad (2) \end{aligned}$$

The *confDIT* property assures that whenever the robots and carts are configured, this configuration allows DIT processing. When the control has just finished a reconfiguration, i.e. $Control = EndReconf$ then *confCorrect* assured that this configuration has not assigned a task to a robot it cannot accomplish due to a failure of a tool. All functional properties were proven under the assumption of *confDIT* and *confCorrect* [GOR06].

There exist several approaches for formal modeling of agent oriented systems e.g. [CP06, Gor00]. Although our system is not an agent-oriented one, it fulfills some of the characterizing aspects of those [Bus98]. Nevertheless these papers do not focus on failure tolerance. A self-optimizing agent oriented system based on load balancing is presented in [BS00].

### 3.3 A Logic for Proving Self-x Properties

In the special case of our production cell the formalization was done by traditional techniques and by hand. For further examples the formalization, especially of the self-x properties, has to be done by hand again. There are two ways to improve the formalization process. The first is a generic translation of the system which results in a lot of explicit formulae. The model can be hard to read for the verifier, what makes error detection difficult. The second is to develop a calculus and a logic with special operators for expressing self-x properties and preserving the structure of the system. As an advantage the formalization of the system would be more readable and more intuitive for the verifier. Especially for interactive verification it would be more comfortable. For example, in the case study above a desired property is: *"the cell always produces correctly unless a reconfiguration takes place"*. In a logic for self-x this property could have a simple form like

$$CELL \models_{RECONF} \Box\, PROD \tag{3}$$

This could be read as: "the system *CELL* fulfills ($\models$"), that it is always in production mode *PROD* unless a reconfiguration (*RECONF*) takes place".

Another example for using an operator is to express that the cell will produce as long as there are enough workpieces available and a reconfiguration phase can provide self-healing. But also properties like "the cell is producing until x happens" or "after n program steps the reconfiguration process is finished and the cell produces again" are interesting.

The system model of the production cell is given as a transition system. The examination of the case study showed that a temporal logic for describing the proof obligations is suggestive. In this case the proof obligations were expressed in CTL [EMCJ99] . But also a linear temporal logic is conceivable. For systems with infinitely many states, where model checking has its limits an interactive verification approach should be also considered.

## 4 Safety Analysis

For safety analysis, fault tree analysis (FTA) is used widely. Nevertheless it is unclear how to formulate a fault tree for an adaptive system. The same problem exists with other more traditional forms of safety analysis like FMEA (failure modes and effect analysis), DCCA and GSPN (generalized stochastic petri-nets) as the system may recover from an occurred hazard. The real hazard is occurring if there is no more possibility for recovery due to too many failing components.

### 4.1 Deductive Cause Consequence Analysis

The deductive cause consequence analysis (DCCA) [ORS05a] finds the relationship between failure as causes and hazards as consequences thereof. Its formalization guarantees

that the causes always happened before the hazards.

To conduct a DCCA we use a finite set $\Delta$ of failure modes $\delta$. The complete list of failure modes can be found with failure sensitive specification [ORS05a] or techniques like HazOp [Kle86]. Assuming a hazard is marked as predicate $H$, we define a critical set $\Gamma \subseteq \Delta$ for a System $SYS$ via the following CTL formula:

$$SYS \models \mathbf{E}\,(\overline{\lambda}\ \mathbf{until}\ H)\ \ where\ \overline{\lambda} := \bigwedge_{\delta \in (\Delta \setminus \Gamma)} \neg\,\delta \tag{4}$$

This formula states that a set of failure modes $\Gamma$ is critical if there exists a run of the system on which no other failure modes than those in $\Gamma$ occur until the hazard appears.

$\Gamma$ is a minimal critical set if no proper subset is critical. Criticality is monotonic wrt. set inclusion. If the empty set is critical, then the system is functionally incorrect, that is even if no failure is present, the hazard may occur. The completeness theorem of DCCA states that if at least one failure from any minimal critical set can be avoided, then the hazard $H$ can not occur [ORS05a]. The resulting minimal critical sets are at least as good as the resulting minimal cut sets from a FTA and can better even better although the fault tree might be proven complete, i.e. FTA can be too pessimistic [ORS05b].

To conduct DCCA, one can exploit the monotonicity of critical sets, i.e. if a set $\Gamma'$ has a critical subset $\Gamma$ then $\Gamma'$ is also critical. Therefore it is feasible to start DCCA from guessed sets or from results of FTA and checking all possible subsets of $\Delta$ is not necessary.

### 4.2 DCCA for adaptive systems

We are working on extending the DCCA approach to be able to cope with adaptive systems. For this we must change the proof obligation of DCCA to support recovery from hazards. The formula (4) does not correctly define a critical set, as recovery may be possible and only if the hazard stays permanently, a critical set is found. An extension of DCCA will then require a proof of monotonicity and a proof of a completeness theorem.

If feasible, an extended DCCA for adaptive systems can provide a possibility for qualitative analysis, i.e. identifying the minimal critical sets and for quantitative analysis, i.e. measuring the occurrence probability of the minimal critical sets.

The result of an extended DCCA may also be used to construct a ”flat” fault-tree for the analyzed system. This may be used directly for quantitative analysis or by transformation to a GSPN [BBD04].

## 5 Descriptive Modeling

Besides formal modeling and verification there is an adequate development process and design method needed for safe and reliable Organic Computing applications, too.

Computer systems can be classified in traditional systems as already known in the last decades, embedded systems which were an integral part of the research in the past few years and Organic Computing systems which have now moved in the center of interest. Many ways to design traditional systems exist already. The Unified Process, the V-Model and a lot more processes are well known software development methodologies and also modeling languages like UML exist for traditional systems. Embedded systems are more complex because of their distribution. They are normally spread over a certain area. So methodologies are needed which can handle this distribution. The newly revised V-Model XT [KBS06] is already directed to analyze and design such systems. There are already tools and methods (UML [Gro06], ROOM [BS94], MARMOT [FG04]) which support the software design of embedded systems with their special properties. Our attention lies on the designing of Organic Computing systems which the existing design-approaches do not support and therefore these approaches must be enhanced or a new development methodology evolved.

The systematic refinement of the enhanced development models should lead in the end to the ability to control the whole system. As however in Organic Computing systems system states might occur which could not be foreseen by the developer - which is contrary to the original top down design - the documents and models must in this place be enhanced by constraints or assertions. The idea is that if the self-x properties act only in this scope - which is defined by these constraints - the system will always meet the essential requirements as long as possible although the environment and tasks change or some system components fail. When a constraint violation occurs, the possibility of interaction with and the controlling of the system is given.

There exists already a couple of approaches for modeling Organic Computing systems and their self-x properties. Tichy, Schilling and Giese [MT04] have designed an approach for self-managing dependable systems with UML and Fault Tolerance Patterns.

Kasinger and Bauer [Kas05] are modeling Organic Computing systems by using a model driven architecture and an extended version of UML 2.0. With this approach they can produce models for each system layer in the analysis and the design phase.

## 6 Conclusion

We presented our case study which we used for initial formal modeling of an Organic Computing system. At the moment we are working on safety analysis based on this formal model. We plan to extend DCCA to adaptive systems for this purpose.

To be able to model bigger examples we work on developing an extension to temporal logic for supporting Organic Computing paradigms. Additional work will be on developing descriptive modeling techniques for Organic Computing systems.

The changing structure of the system, inherent to Organic Computing systems, is the challenge for formal modeling techniques, even without taking emergent behavior into consideration. The next steps will be finishing work on safety analysis and to abstract the findings from the case study to more general forms of self-adaptive systems.

## References


[BBD04] S. Bernardi, A. Bobbio, and S. Donatelli. Petri Nets and Dependability. *Lectures on Concurrency and Petri Nets: Advances in Petri Nets*, pages 125–179, June 2004.

[BS94] G. Gullekson B. Selic, P. T. Ward. *Real-Time Object Oriented Modeling*. John Wiley and Sons, 1994.

[BS00] S. Bussmann and K. Schild. Self-Organizing Manufacturing Control: An Industrial Application of Agent Technology, 2000.

[Bus98] S. Bussmann. Agent-Oriented Programming of Manufacturing Control Tasks, 1998.

[CP06] P. Engrand C. Pecheur, R. Simmons. Formal Verification of Autonomy Models: From Livingstone to SMV. In *Agent Technology from a Formal Perspective*, pages 103–113. Springer Verlag, 2006.

[EMCJ99] D. A. Peled E. M. Clarke Jr., O. Grumberg. *Model Checking*. The MIT Press, 1999.

[FG04] Frauenhofer-Gesellschaft. Method for Component-Based Real-Time Object-Oriented Development and Testing, 2004.

[Gor00] D. F. Gordon. APT Agents: Agents That Are Adaptive, Predictable and Timely. In *Proceedings of the First Goddard Workshop on Formal Approaches to Agent-Based Systems (FAABS'00)*, 2000.

[GOR06] M. Güdemann, F. Ortmeier, and W. Reif. Formal Modeling and Verification of Systems with Self-x Properties. In *Autonomic and Trusted Computing 2006, Proceedings*. Springer LNCS, 2006.

[Gro06] OMG Object Modeling Group. The Unified Modeling Language, 2006.

[Kas05] H. Kasinger. Ein MDA-basierter Ansatz zur Entwicklung von Organic Computing Systemen (in German). Technical report, University of Augsburg, 2005.

[KBS06] Koordinierungs-und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung KBSt. Das neue V-Modell XT Release 1.2 - Der Entwicklungsstandard für IT-Systeme des Bundes (in German), 2006.

[Kle86] T. A. Kletz. Hazop and HAZAN Notes on the Identification and Assessment of Hazards. Technical report, Inst. of Chemical Engineers, Rugby, England, 1986.

[McM90] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1990.

[MH95] J. McLean and C. Heitmeyer. High assurance computer systems: A research agenda, 1995.

[MMB03] A. Montresor, H. Meling, and O. Babaoglu. Towards Self-Organizing, Self-Repairing and Resilient Large-Scale Distributed Systems. In *Future Directions in Distributed Computing - Research and Position Papers*, pages 119–123, Bologna, Italy, 2003. Springer.

[MT04] H. Giese M. Tichy, D. Schilling. Design of Self-Managing Dependable Systems with UML and Fault Tolerance Patterns, 2004.

[ORS05a] F. Ortmeier, W. Reif, and G. Schellhorn. Deductive Cause-Consequence Analysis (DCCA). In *Proceedings of IFAC World Congress*, 2005.

[ORS05b] F. Ortmeier, W. Reif, and G. Schellhorn. Formal safety analysis of a radio-based railroad crossing using Deductive Cause-Consequence Analysis (DCCA). In *Proceedings of European Dependable Computing Conference EDCC*, 2005.