Formal modeling and verification of systems with self-x properties

Matthias Güdemann, Frank Ortmeier, Wolfgang Reif

Angaben zur Veröffentlichung / Publication details:

Güdemann, Matthias, Frank Ortmeier, and Wolfgang Reif. 2006. "Formal modeling and verification of systems with self-x properties." In *Autonomic and Trusted Computing: Third International Conference, ATC 2006, Wuhan, China, September 3-6, 2006*, edited by Laurence T. Yang, Hai Jin, Jianhua Ma, and Theo Ungerer, 38–47. Berlin: Springer. https://doi.org/10.1007/11839569_4.

Formal Modeling and Verification of Systems with Self-x Properties*

Matthias Güdemann, Frank Ortmeier, and Wolfgang Reif

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

Abstract. In this paper we present a case study in formal modeling and verification of systems with self-x properties. The example is a flexible robot production cell reacting to system failures and changing goals. The self-x mechanisms make the system more flexible and robust but endanger its functional correctness or other quality guarantees. We show how to verify such adaptive systems with a "restore-invariant" approach.

1 Introduction

Still today, many technical systems are tailored very rigidly to the originally intended behaviour and the specific environment they will work in. This sometimes causes problems when unexpected things happen. A possible way to make such systems more dependable and failure tolerant is to build redundancy in many components. Another approach is to design systems from the beginning in such a way, that they can dynamically self-adapt to their environment. The benefit of these adaptive systems is that they can be much more dependable than conventional systems, without increasing system complexity too much, as not every scenario must be modeled explicitly.

From the point of view of formal methods, these systems are more difficult to describe as their structure may change with the adaption. This can lead to problems when functional correctness of such a system is to be proven. Nevertheless, in many safety critical fields like production automation, avionics, automotive etc., functional correctness and quality guarantees are crucial. We show how an adaptive system can be modeled with formal methods and its functional correctness be proven under adaption. We illustrate this technique with a case study from production automation.

2 Case Study

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

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

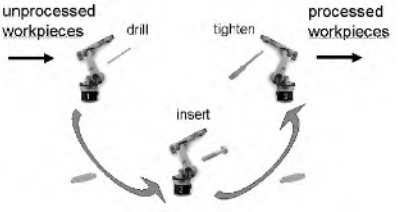

Fig. 1. Valid configuration of robot cell

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. This situation is shown in fig. 1.

2.2 Self-organization

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.

As the robots can switch tools 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 can be resolved as shown in fig. 3. Now the left robot drills, the right robot 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 the direction of the incoming and outgoing

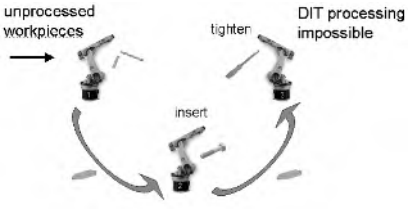

Fig. 2. Hazard due to broken drill

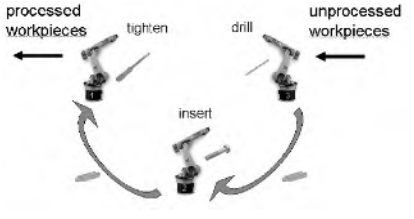

Fig. 3. Reconfigured robot cell

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

2.3 Self-adaption and Self-optimization

Another form of adaption is possible in the production cell when partially processed workpieces arrive. A RFID tag can be used on the workpieces that indicates whether they have already a drilled hole. Those that have the hole drilled do not have to be processed by the robot assigned the drill task. If there is an additional transport cart available, then it can bring the partially processed workpieces directly to the robot that inserts the screws.

When this has happened, the robot that has been assigned the task to insert the screws might become the bottleneck in the system. If an additional robot and cart are available, they can be integrated to self-optimize the throughput of the production cell.

Again, self-x properties are only possible because of internal redundancy in the system. This includes redundant tools that can be switched and redundant robots or transport carts. The difference to traditional redundant systems is that they are dynamically configured and may be used for other tasks if not needed at the moment.

The production cell is also capable to self-adapt using *graceful degradation* to fulfill at least parts of its functions as long as possible. In this example this could be drilling holes in a workpiece and inserting screws but not tighten them. Yet another form would be to use one robot to accomplish more than one task. This is also a case of *graceful degradation* because it preserves the functionality but diminishes the throughput of the production cell considerably.

Although small, the example exhibits several aspects of self-x properties and especially self-adaption. Nevertheless, several interesting questions arise: How does the dynamically changing organization of the production cell affect functional correctness? What happens while a reconfiguration takes place? Does the system produce correctly after a reconfiguration?

3 Formal Model

When trying to build a formal model for an adaptive system, the question arises how to represent the dynamically changing characteristics of such a system. We found that these can be modeled with techniques similar to conventional systems. We used transition automata as representation for the robots, the carts, the workpieces and the reconfiguration control. The functional properties of the system can be expressed using temporal logic formulas.

The crucial point of the modeling is the treatment of the reconfiguration. We regard a run of a system as separated in production phases and reconfiguration phases. A production phase is characterized by the invariant “The system configuration allows for processing a workpiece in DIT order”. The end of a production phase is marked by the violation of this invariant. The purpose of the

reconfiguration phase is the restoration of the invariant. When this is achieved, the reconfiguration phase is over and a new production phase starts. We use the term "restore-invariant" for this approach.

We did not implement a specific reconfiguration algorithm in the model but only specified it in a "top-down" way, as restoration of a functional invariant. This technique can also be applied to the other mentioned self-x properties. The crucial point here is that the algorithm must be able to decide whether these invariants hold and restore them if not. Seeing reconfiguration in this abstract way gives us the advantage that it is sufficient to prove that an algorithm can restore the invariant to show that the algorithm provides correct reconfiguration, thus modularizing our model.

We implemented the formal model of the adaptive production cell as transition system in the SMV model checker [7]. This formalization allows specification of functional correctness in CTL (computational tree logic) and LTL (linear time logic), or their semantics see[3].

3.1 Transition Systems

Due to space restrictions not all transition systems are shown for the automata. The respective transition preconditions are explained in the text. Dashed lines indicate the effect of an interrupting reconfiguration. If used, this confines reconfiguration from normal functioning.

Control Transition System. The Control performs the reconfiguration of the production cell. Its transition system is shown in fig. 4. It waits in state $Reconf$ until all robots and carts are in their respective reconfiguration states. Then it enters the state $Initialize$. After this, one of the states of the $Robot1Conf$ multi-state is entered, then one of the $Robot2Conf$ states and finally one of the $Robot3Conf$ states. Which one of the states is entered, decides which task is assigned to the corresponding robot. The assignment of the routes to the carts is done analogously in the $Cart1Conf$ and $Cart2Conf$ multi-states. Which task is assigned is chosen indeterministically. Correct assignment for processing workpieces is assured by the specification of the reconfiguration algorithm explained in Sect. 3.3.

Robot Transition Systems. The initial state $Reconf$ is left when the Control assigns a new task to the corresponding robot. The succeeding states are either $readyD$, $readyI$ or $readyT$ for the respective tasks.

When the robot is in $readyD$ state it waits for a new workpiece to arrive. When this happens it enters state $busyD$. If the workpiece has already been processed with the tool the robot uses, it enters directly $doneD$, simulating passing through of the already processed workpiece. After $busyD$ the $doneD$ state is entered indicating that the workpiece processing is complete and the robot waits for a cart that fetches the workpiece. When this happens, the robot enters $readyD$ again. The same holds for the other possible tasks. When a reconfiguration is initiated by the Control, then the robot leaves its current state and reenters $Reconf$.

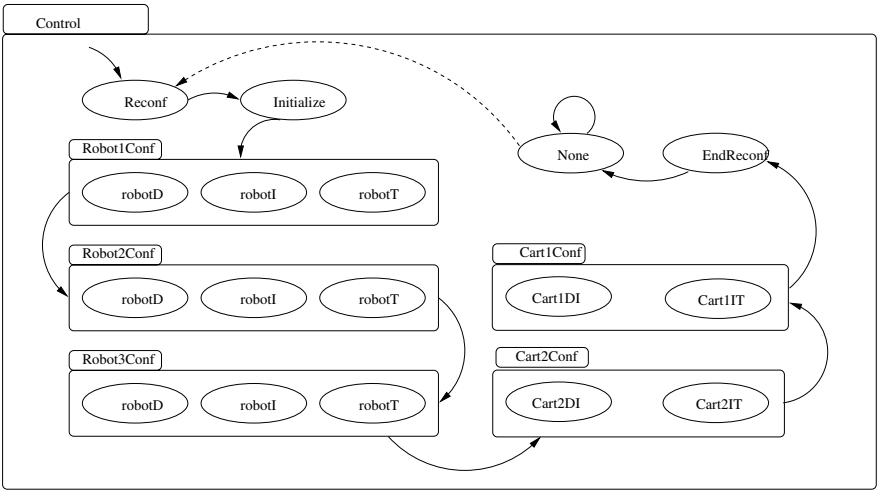

Fig. 4. Control transition system

Workpiece Transition Systems. The workpieces are defined via two automata. The first one is WP_i^{pos}. It describes the position of the workpiece i in the production cell.

The initial state is $Before$ if the cell is configured, then the next state is $infrontD$ as the workpieces are delivered to the driller not with the aid of carts but with a conveyor that is not modeled. When the robot that has been assigned the drill task is ready, then WP_i^{pos} enters the state inD. When the task is done, it enters $behindD$. When the workpiece is fetched, it enters either the state $Cart1$ or $Cart2$, depending on which cart arrived. After the respective cart arrives at either the $infrontI$ or $infontT$ position, the WP_i^{pos} enters the state corresponding to this position. The other tasks are modelled in a similar way, the only difference is, that after $behindT$, the WP_i^{pos} enters the state $Before$ again, instead of being put on a cart.

The second transition system for the workpiece is WP_i^{state}. It indicates which tasks have already been completed on the workpiece. It consists of an 3-bit array, each bit corresponds to one of the possible tasks and has the value 1 if the task has been done and 0 otherwise. When the workpiece leaves the production cell, then WP_i^{state} is reset to its initial state and it is reintroduced into the production cell. When a reconfiguration takes place, the workpieces that are in the production cell are brought to the $Before$ state again, but their completed tasks are preserved.

Cart Transition System. The carts are represented as the product automaton of three different automata. The first one is C_i^{conf}, see fig. 5. Its initial state is $Reconf$. The succeeding state is either DI or IT, depending on the configuration that the Control automaton assigns. If $C_i^{conf} = DI$ then the corresponding

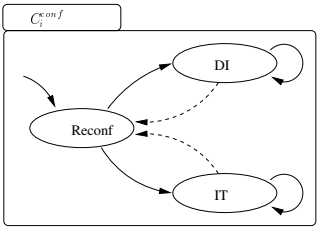

Fig. 5. Cart configuration

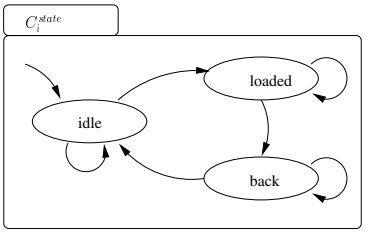

Fig. 6. Cart state

cart is assigned the route between the drilling and the screw-inserting robot, $C_i^{conf} = IT$ is then the route between the screw-inserting robot and the screw-tightening one. When a reconfiguration starts, then C_i^{conf} enters $Reconf$ again, to get a new configuration.

The second automaton for describing a cart is C_i^{state}, see fig. 6. The initial state is *idle* and indicates that the cart is waiting behind a robot and waiting for a workpiece processed by this robot. The state *loaded* is entered when a workpiece has been processed by the robot the cart waits behind, and is to be transported on the assigned route. When the cart arrives at the next robot and the workpiece is fetched, then the state *back* is entered. The *idle* state is reentered when the position of the cart is again behind the robot.

The third automaton for the description of the carts is C_i^{pos}. It represents the position the cart is at. The positions correspond to the possible positions of the workpieces. The initial state is $Undefined$. Its route is abstracted to three states, behind a robot, between two robots and in front of the next robot. Depending on the assigned configuration these are either the drilling and inserting or the inserting and tightening robot.

Failure Automata. For the modeling of failures we use failure automata. These can be either *transient* or *persistent* see fig. 7. The initial state of a failure automaton is *no*, i.e. there is no error at the moment. The automaton can indeterministically enter state *yes*, indicating that an error has occurred. A *transient* failure can disappear, again in an indeterministic way.

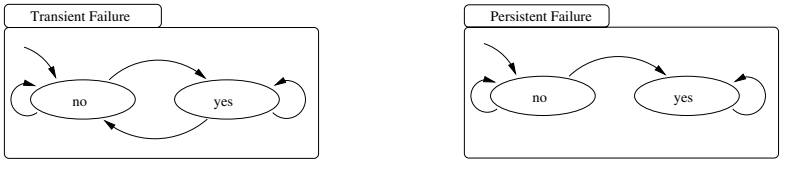

Fig. 7. Failure automata for transient and persistent failures

3.2 Predicates

For the predicates we use the notion $A = s$ as abbreviation for the predicate "automaton A is in state s". For the formal model we define the predicates R_i^a with $i \in \{1,2,3\}$ and $a \in \{d,i,t\}$. These variables are true if robot i has been assigned task a by checking whether the corresponding automaton is in one of the states corresponding to task a. Together with the configuration automata of the carts we define:

$$
\begin{aligned}
robotConf &:= (R_1^d \vee R_1^i \vee R_1^t) \wedge (R_2^d \vee R_2^i \vee R_2^t) \wedge (R_3^d \vee R_3^i \vee R_3^t) \\
cartConf &:= (C_1^{conf} \neq Reconf) \wedge (C_2^{conf} \neq Reconf) \\
conf &:= robotConf \wedge cartConf \\
ditCapable &:= \bigwedge_{a \in \{d,i,t\}} \Big(\bigvee_{j \in \{1,2,3\}} R_j^a \wedge \Big(\bigwedge_{k \in \{1,2,3\} \setminus j} \neg R_k^a \Big) \Big) \\
cartCapable &:= (C_1^{conf} \neq C_2^{conf}) \wedge (C_1^{conf} \neq Reconf) \wedge (C_2^{conf} \neq Reconf)
\end{aligned}
$$

This means that $robotConf$ holds when all robots have a task assigned. The same holds for the carts with $cartConf$. The variable $ditCapable$ is true when the assignment of tasks to robots includes all three tasks. As the variables R_i^a are defined via the states of the robot automata, the formula $R_i^a \rightarrow \bigwedge_{b \in \{d,i,t\} \setminus a} \neg R_i^b$ always holds.

To model broken tools as failures in the model of the production cell we define transient failure automata as explained in Sect.3.1 for all tools of all robots. A complete list of possible failure modes can be found with techniques like failure-sensitive specification [8] or HazOp [5]. These failure automata are called $fails_i^a$ with $i \in \{1,2,3\}$ and $a \in \{d,i,t\}$. Using these automata we define additional boolean variables:

$$
\begin{aligned}
ditFailure_j &:= \bigvee_{a \in \{d,i,t\}} (R_j^a \wedge (fails_j^a = yes)) \\
ditFailure &:= \bigvee_{j \in \{1,2,3\}} ditFailure_j
\end{aligned}
$$

This means that $ditFailure_j$ holds if robot j has been assigned a task it cannot perform as the corresponding tool is broken and $ditFailure$ indicates that one or more robots have been assigned a task that is impossible at the moment. Whenever the external Control detects that $ditFailure$ holds, then a reconfiguration is triggered.

For proving functional properties and specifying a correct reconfiguration algorithm the predicate $ditPossible$ is needed that holds if a correct configuration is still theoretically possible. We used the disjunction of all correct robot configurations for this. It is important to mention, that this is not needed in the model itself but only to specify the reconfiguration algorithm and to prove functional correctness. That means that $ditPossible$ may also be defined in another way, e.g. to model *graceful degradation* adaption.

3.3 Specification of Reconfiguration

For this specification of the reconfiguration we used LTL. SMV allows LTL formulas in assumed properties. The two specifications are as follows:

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

That is, $confDIT$ specifies that every reconfiguration results in a sensible configuration of the production cell. Every tool must be available and the carts have distinct routes assigned. The property $confCorrect$ specifies that whenever a reconfiguration has just been finished, $ditFailure$ is false as long as correct configuration is still possible. All functional properties in the next section are proven under the assumption that $confDIT$ and $confCorrect$ hold.

4 Verification

We see a run of the system divided in phases of production and reconfiguration. In a production phase, workpieces are processed in a straightforward way. When a tool breaks, then a reconfiguration takes place that changes the organization of the cell. The propositions we want to prove are the same as in a conventional system, i.e. that processing of the workpieces is done in correct DIT order and on all workpieces that enter the production cell.

Proposition 1 assures that as long as $ditPossible$ holds every workpiece will finally be processed by all tools. Therefore, as long as processing is theoretically possible, all tasks are executed on all the workpieces.

$$\mathbf{AG}\,(ditPossible \rightarrow (\,\mathbf{AF}\,WP_i^{state} = [1,1,1])) \tag{1}$$

This does not yet guarantee that processing is done in the correct DIT order. For this property we prove the following:

$$\begin{aligned} &\mathbf{AG}\,WP_i^{state} = [0,0,0] \vee \\ &\mathbf{A}\,[WP_i^{state} = [0,0,0]\,\mathbf{until}\,WP_i^{state} = [1,0,0] \vee \\ &(\,\mathbf{A}\,[WP_i^{state} = [1,0,0]\,\mathbf{until}\,WP_i^{state} = [1,1,0] \vee \\ &(\,\mathbf{A}\,[WP_i^{state} = [1,1,0]\,\mathbf{until}\,WP_i^{state} = [1,1,1]])])] \end{aligned} \tag{2}$$

Proposition 2 proves that processing is never done in a wrong order. Together with proposition 1 we know that as long as processing is possible, processing with all three tools is done.

The next propositions show that the modeling of our cell is sensible Proposition 3 shows that workpieces never occupy the same position "in" a robot.

Proposition 4 shows that carts are never at the same position. Their position is not tracked in a reconfiguration phase. Proposition 5 shows that if for a cart

is loaded, then it carries a workpiece, i.e. a workpiece has the same position as the cart.

$$\mathbf{AG}\,(WP_i^{pos} = WP_j^{pos} \rightarrow WP_i^{pos} \notin \{inD, inI, inT\}) \tag{3}$$

$$\mathbf{AG}\,((C_i^{conf} \neq Reconf \wedge C_j^{conf} \neq Reconf) \rightarrow (C_i^{pos} \neq C_j^{pos})) \tag{4}$$

$$\mathbf{AG}\,(configured \rightarrow (C_i^{state} = loaded \rightarrow \exists j : WP_j^{pos} = C_i^{pos})) \tag{5}$$

These proposition show that the modeling of the production cell is correct according to the requirement that failures can occur and processing is correctly adapted to these failures as long as possible. Furthermore we provided several propositions that showed that the cell is modeled in a sensible way beyond the adaption capabilities.

5 Related Work

Much of the available work on adaptive systems and verification thereof is based on agent-oriented programming. These mentioned papers have no similar concept to our "restore-invariant" technique for top-down design for systems with self-x properties.

Bussman et al. [1] define several functional requirements and software engineering requirements that an agent-oriented system must fulfill in order to be qualified for industrial control problems. Both functional and software engineering requirements are met by our model of the production cell, although it was not designed having agent-orientation in mind.

In [2] Bussmann et al. describe an adaptive production system based on agent-technology. An auction based approach is taken where agents bid for tasks and a self-organization of material flow takes place. They prove that their approach is free of deadlocks and give good reasons and empirical observations for increased productivity. This differs from our approach as it is more directed to increasing throughput and not to make the system more dependable to failure of components.

Kiriakidis and Gordon-Spears describe in [4] an approach to restore supervision and assure specified behaviour of robot teams. It it based on transition automata and a language that is expressed by these automata. This language is changed by learning algorithms triggered by unforeseen events. This approach is directed more to team composition instead of robot functionality as in our example.

Another approach is taken by Cornejo et al. in [6]. A dynamic reconfiguration protocol is specified and verified using the LOTOS language. It consists of a configurator agent and application agents that communicate over a software bus. Its asynchronous communication would be an interesting way to implement a reconfguration algorithm that fulfills the mentioned invariant.

6 Conclusion

We presented a case study in formal modeling and verification of self-adaptive systems. The example production cell exhibits several self-x properties, particularly self-organization.

We have shown a way how to guarantee functional correctness under the presence of failures and reconfiguration. The idea is to impose invariants to be maintained by the system. In case of failures, the invariant is violated and the task of the reconfiguration or the adaption is to restore these invariants. This leads to a "top-down" design approach for self-adaptive systems, separating specifications from their implementations. We call this approach "restore-invariant".

In the example we focused on functional correctness but the invariants could also express other goals like throughput performance, load-balancing or graceful degradation.

The next step is to look at the additional self-x properties of the production cell mentioned in Sect. 2.3. Other interesting topics are overcoming the limitations of finite state spaces and measuring the benefit of adaptive systems. Of course we will also generalize the concepts from this case study to make this approach applicable to general forms of self-adaptive systems.

References

[1] S. Bussmann. Agent-oriented programming of manufacturing control tasks, 1998.
[2] S. Bussmann and K. Schild. Self-organizing manufacturing control: An industrial application of agent technology, 2000.
[3] Doron A. Peled Edmund M. Clarke Jr., Orna Grumberg. *Model Checking*. The MIT Press, 1999.
[4] K. Kiriakidis and D. F. Gordon-Spears. Formal modeling and supervisory control of reconfigurable robot teams. In *FAABS*, pages 92–102, 2002.
[5] T. A. Kletz. Hazop and HAZAN notes on the identification and assessment of hazards. Technical report, Inst. of Chemical Engineers, Rugby, England, 1986.
[6] R. Mateescu M. A. Cornejo, H. Garavel and N. De Palma. Specification and verification of a dynamic reconfiguration protocol for agent-based applications. In *Proc. of the IFIP TC6*, pages 229–244, Deventer, The Netherlands, The Netherlands, 2001. Kluwer, B.V.
[7] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1990.
[8] F. Ortmeier and W. Reif. Failure-sensitive specification: A formal method for finding failure modes. Technical Report 3, Institut für Informatik, Universität Augsburg, 2004.