

Systematic Verification of Flexible Manufacturing Control Software

Christian Brecher, Tilman Buchner, Andreas Karlberger

Abstract—Increasing the performance and flexibility of automated manufacturing systems is a key success factor for today's production companies. Flexible manufacturing systems (FMS) have proven to be particularly suitable in this regard since they support small lot sizes and high numbers of variants at the same time. Among the most important problems of FMS are the huge expenditure of time and the high costs for engineering its control software. Engineering in this context refers to all aspects from planning the specific production process, assigning control programs to machines, implementing software modules, to testing the whole configuration. In this paper, we describe a new approach to verify a distributed agent-oriented shop floor control system systematically on the basis of a virtual process qualification.

Index Terms— Distributed control systems, intelligent fault detection and identification, software agents for intelligent control systems, flexible manufacturing systems

I. INTRODUCTION

The international competitive environment results in an essential change in production engineering. Small lot sizes and high numbers of variants in combination with shorter product life cycles demand - particularly in the area of discrete manufacturing - flexible automation solutions [1]. Thereby flexible manufacturing systems (FMS) or flexible manufacturing cells (FMC) are being applied preferentially. FMS are fully- or semi-automated production systems, consisting of one or more machine tools, an automated material- and storage system and one or more loading- and unloading stations. The typical configuration of such a system conforms to a by-pass or network structure.

At the center of a FMS a storage system is located and single machine tools are connected to its longitudinal sides where they can be fed automatically, see Fig. 1. This kind of flexible production system makes an economic manufacturing process

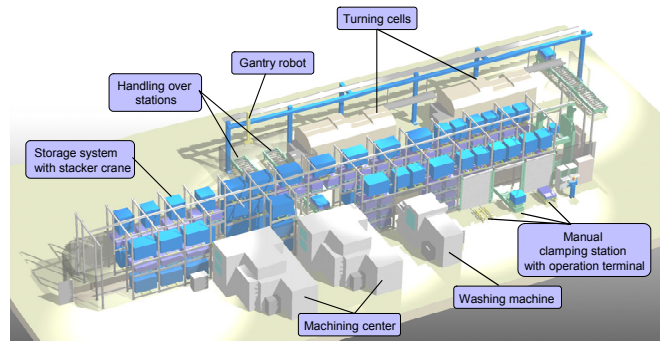


Fig. 1. Flexible manufacturing system for chipping [source: FASTEMS]

under dynamic market conditions possible [4]. Basically FMS can be classified into different application domains such as milling and turning, sheet metal processing, and sawing of slug material [2].

The more production has to respond to varying batch sizes or product variants, requirements on the *sequence control* of the system increases. The *sequence control* comprises control operations following a mandatory, iterative progression whereby the routing from one operation to the next results in dependence on a precondition (IF-THEN) [3]. As a running example, we consider the production of a "Connect 4 Game". The production process includes sawing, milling and assembly operations. For that, the sawing machine KASTO and the milling machine MAHO need to cooperate. In addition, the robotic palletizer ROB_GANTRY and the transportation system AGV are responsible for the material flow. Fig. 2 describes the process planning for the sawing machine KASTO in detail. As the figures outlines, the KASTO will request the transportation system AGV to deliver a pallet to store sawed raw material as well as the robot palletizer to be prepared for picking the sawed workpieces. Since these are two independent operations, they can be executed simultaneously.

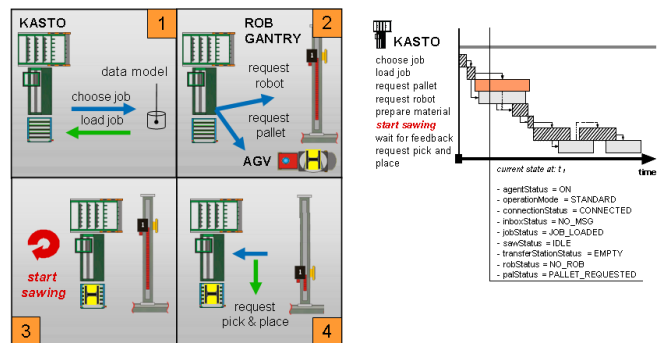


Fig. 2. Sequence control exemplified by a sawing cell

Manuscript received July 21, 2007. The work has been supported by the German Federal Ministry of Education and Research (BMBF) within the research project "Ramp-Up/2".

Prof. Dr.-Ing. Christian Brecher, is with the Laboratory for Machine Tools and Production Engineering (WZL), RWTH Aachen University, 52074 Aachen, Germany, (e-mail: c.brecher@wzl.rwth-aachen.de).

Dipl.-Ing. Tilman Buchner, is with the Laboratory for Machine Tools and Production Engineering (WZL), RWTH Aachen University, 52074 Aachen, Germany, (e-mail: t.buchner@wzl.rwth-aachen.de).

Dipl.-Ing. Andreas Karlberger, is with the Laboratory for Machine Tools and Production Engineering (WZL), RWTH Aachen University, 52074 Aachen, Germany, (e-mail: a.karlberger@wzl.rwth-aachen.de).

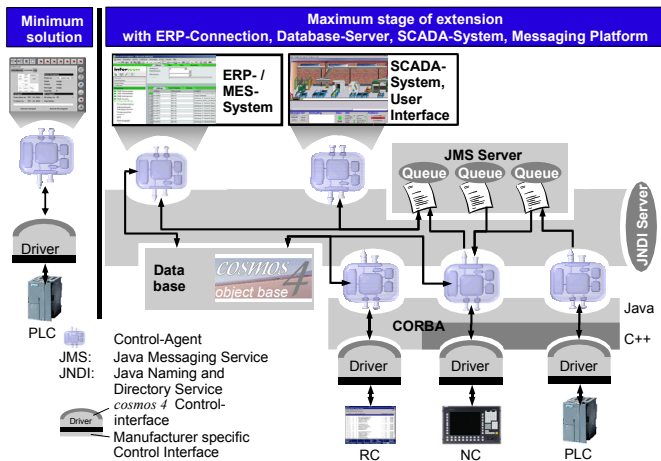


Fig. 3. System architecture of the *cosmos 4* platform

As soon as the KASTO receives the commitments from the AGV and the robotic palletizer, it will start the sawing process.

This example clarifies that to manufacture a complex workpiece consisting of several successive process operations, which are coupled by clamping, handling and transportation operations, the control- and coordination effort increases continuously.

The function of the FMS control software is to guide this semi- or full automated production process. To realize FMS control software there exists central as well as locally control architectures. The limited availability of productions system in the 80s using central control architectures led to the development of modular, locally architectures. Next to this evolution, there is an obvious trend to vertical integration of different information systems in production industry by the keyword MES (manufacturing execution system). In a nutshell state-of-the-art FMS control software requires locally, generic, component-based architectures consisting of single software elements operating autonomously.

II. FMS CONTROL SOFTWARE

To overcome this challenge, a new automation solution, the *cosmos 4* platform, has been developed at RWTH Aachen University's Laboratory for Machine Tools and Production Engineering (WZL) [4]. It is a PC-based, distributed, agent-oriented approach to universal and open control components for different types of FMS. The *cosmos 4* platform addresses the heterogeneity of FMS by two major features.

Firstly, it includes an independent reference model that holistically fits the static structure of FMS and its components as well as the dynamic behavior of FMS operations. Secondly, it provides a new advanced concept for increased "programmability" of control components (agents) reflecting that the control logic is at the core of the FMS control system's software. This multi-agent system comprises agents with programmable control logic which will be run through cyclic as well as event-driven and with open interfaces for the integration of functional units. Furthermore, a flexible message-oriented communication platform is provided, as well as a service-oriented interface to connect different device

controllers, see Fig. 3. Thereby, agents are able to coordinate processes in FMS stand-alone. An agent can be applied to control a single machine tool up to a complete FMS cell. During the job processing the agents can communicate with each other for the purpose of e.g. material or handling supply. These features ease the adaptation to changing requirements. But due to its vast applicability and scalability, the profitability of *cosmos 4* depends predominantly on its engineering costs.

III. VIRTUAL ENGINEERING

Today, the adaptation and reconfiguration of FMS control software to a specific production system or customer requirements is associated with programming work. The interventions and changes in the existing software modules as well as in the data model of the control software lead to huge expenditures of time. In addition, a largely manual implementation results in high error quota. Depending on the scope of modifications the effort of testing and placing the FMS control software into operation increases.

The consequence is that costs in the double-digit percentage range of the investment costs of an automated manufacturing system arise [4]. This problem traces back to the lack of concept and proper tool support for engineering PC-based FMS control software. Engineering here includes all aspects from planning the concrete production process, to assigning machines to control programs, to implementing new software modules, and to testing the operability and capability of the whole configuration.

Therefore, a new concept called "Systematic Engineering for Flexible Manufacturing Control Software" has been developed [8]. The motivation to develop a comprehensive engineering concept is based, besides the obvious target to reduce the engineering cost, on the general ambition to shorten the ramp-up time of production systems and their control software significantly. Systematic engineering can be understood as a process, which enables the adaptation (reconfiguration) as well as the optimization of an automated production system over its entire life cycle. The general process follows the well-known waterfall software development model and is subdivided into eight levels, see Fig. 4. Each single step will be run through partly concurrently and partly iteratively and not necessarily in a linear sequence.

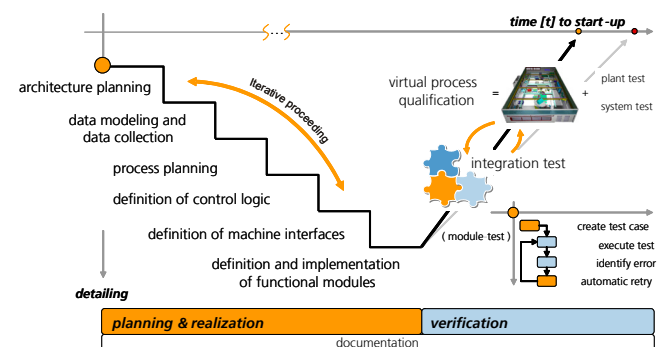


Fig. 4. Approach to systematic engineering

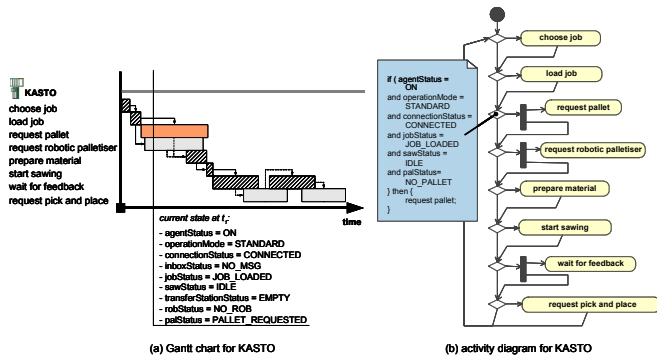


Fig. 5. Gantt chart and activity diagram example for KASTO

In principle, the engineering concept can be structured into two phases, the planning and realization and the verification phase. Firstly the planning and realization phase will be outlined shortly; subsequently to this the verification phase will be presented in detail.

IV. PLANNING AND REALIZATION PHASE

The output of the planning and realization phase can be divided into two categories: the engineered object- and the engineered behavior model. The static object model comprises machine-, workpiece holder-, workpiece-, tool-, manufacturing-, material- flow- as well as personal-related objects and is used to describe the structure of a FMS [6]. On the other hand, the behavior model defines the sequence (control) of the FMS. Basic principle of the engineered “objects” and “behavior” is a model driven engineering approach (MDA).

Fundamental idea of this approach is to subdivide FMS control software into three parts: a generic part, keeping identical, a schematic part being different for each realization but of the same systematic, and one specific part being individual and not generalizable. The generic part represents the *cosmos 4 platform* itself, the object-oriented reference model as well as the programmable control components. The schematic part comprises repeating code segments e.g. the *control logic of an agent* which always has the same fundamental design. In contrast to this, individual code segments like capability modules or machine drivers

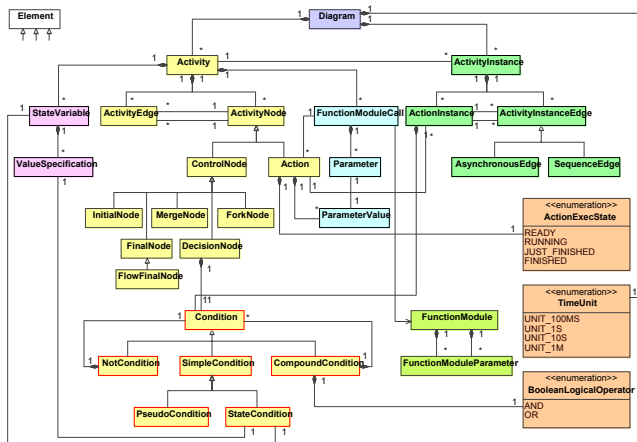


Fig. 6. Metamodel for Gantt charts and activity diagrams

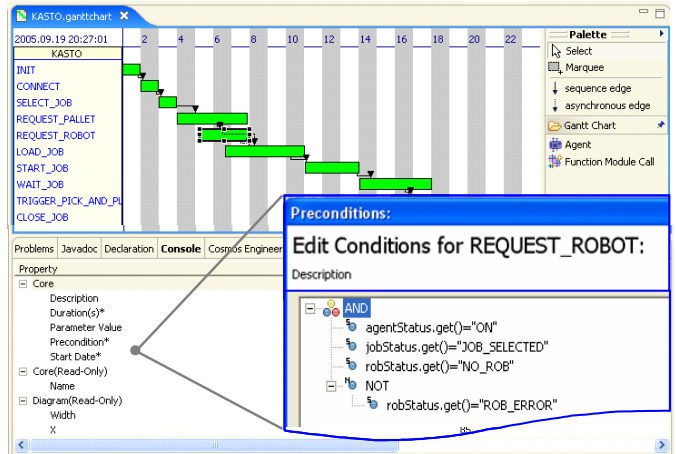


Fig. 7. *cosmos 4e*: overview of Gantt chart and condition editor

have to be implemented individually concerning a custom-designed production system.

The idea of the MDA approach is to derive a model out of the schematic part of the software. Instead of manual implementing the source code, the user is able to refine and substantiate e.g. the behavior of a control component in a model. Through out the model information the specific control logic can be automatically generated and processed by the corresponding agent [5].

In the following the MDA approach will be illustrated on the example of a behavioral model. The key idea of the concept is as well applicable to the object model but out of scope of this paper. The behavior model is used to describe the sequence control of a certain control component according to a state machine. For this purpose, two different views, a Gantt chart, representing the sequence planning, and an activity diagram, to specify the transition conditions, has been chosen, see Fig. 5. Precondition to mapping both views in one model is a common metamodel. Therefore, a metamodel which is based on an extended UML activity diagram to fit the special requirements of the *cosmos 4 control* software has been developed and realized on the basis of the Eclipse Modelling Framework (EMF), see Fig. 6.

Integrated into a prototype, called *cosmos 4e* (e for Engineering), which is created as an Eclipse plug-in, the user is able to create an agent, to specify its behavior control as well as to parameterize its functional modules [5]. Fig. 7 illustrates the general procedure how to model the dynamic behavior of a control component on the example of the sawing machine KASTO in detail. Firstly, the user has to arrange the function blocks e.g. “SELECT_JOB”, “REQUEST_PALLET” etc. in a chronological sequence over time, to specify their input parameters as well as their execution type ((-) synchronous), see Fig. 5 (a). Secondly, he has to bind a corresponding capability module to the function block and to define a transition condition which ensures that the execution is limited to a specific system state of the agent, see Fig. 5 (b). Hereto, a library consisting of predefined capability modules is available. The spectrum of capability modules reaches from interacting with device controllers, to accessing the control software’s data

model as well as to communicating with other agents in order to interact for the purpose of e.g. material or handling supply.

Having completed the model, the user can start the automated code generation and creates the control logic source code which can be processed by the corresponding agent. The generation is preceded by a simple model verification that ensures that the model is syntactically valid.

V. SYSTEMATIC VERIFICATION

After the planning and realization phase an engineered shop floor control software is available to place into operation. All schematic components of the control system are generated from the object or behavior model fully automated. These comprises the components of the FMS as well as the control logic of the agents. The program parts being generated automatically by means of the model-transformation can be considered as formally correct code. In addition, the *cosmos 4 platform* including its communication and data-storage mechanism can be treated as fully functional.

Modelling failures in the object model that can emerge e.g. throughout a wrong assignment of a gripper type to handle a specific workpiece or logically inconsistent transition conditions in the behavior model, which results in failures in the control logic of an agent, cannot be identified by a syntactical check of a model. Nowadays, such failures are only being identified at the real plant and thus lead to high downtimes and costs. Cause for this, is the lack of adequate concepts to assure the quality in the area of control technology.

Thereby the manufacturing equipment vendor has to cope with the dilemma of ensuring the *operability and capability on the one hand and the need for a fast ramp-up on the other hand*.

This paper introduces a method that combines a test systematic and a virtual kinematical shop floor model to enable a "virtual process qualification" and hence merge the two goals. The objective target of the test systematic is to identify modeling errors in the object-model and within the behavior models of the FMS control software.

A. Testing principle

The verification of distributed control software is predominantly characterized by its concurrent processes. Due to its pro-active behavior patterns, narrowing down the error search-space is being significantly complicated by agent-based control software. The interaction of agents as a crucial element of a control system has to be taken into account for the design of a test method. Hereto the principle of interaction in an agent-based control system will be illustrated on the *cosmos 4 control* platform under consideration of a sawing process on the sawing machine KASTO (cp. Fig. 5 (a)).

The agent KASTO responsible for the sequence control of a manufacturing process to saw slug material is booted and starts to process its control logic cyclically. Next the agent's function block *load_job* loads a sawing process (SAW_4WG) from the object model as soon as the ERP system releases a corresponding job. Then the agent updates its state by changing the state variable *job_status* to *JOB_LOADED*. Now the next

transition condition is satisfied and the function block *request_pallet* checks if a proper pallet for storing sawed workpieces is located at the corresponding transfer station at KASTO. In case of an empty station a request message to the agent AGV is being sent. The AGV receives the request, then generates autonomously a corresponding transport job *TRANSP_PAL*. After finishing the transport job the agent AGV informs the agent KASTO. Consequently, the agent KASTO switches its current status *palStatus* to *AVAILABLE*. Here the pallet transportation job has been taken place asynchronously while concurrently calling the function block *request_robot_palletizer*. The influence of the machine behavior on the agent is rather obvious within the *start_sawing* and *wait_for_feedback* functions. Depending on the process outcome of the KASTO's Programming Logic Control (PLC) a standardized return code is returned through a service-based driver interface that again updates the agent's status.

The example illustrates the dependency of control components among each other as well as the associated tight-knit crosslinking of job processes, object model, state variables and function modules. To verify the interplay of these elements the control logic of each single control component has to be considered. Therefore the agent represents the smallest unit to be tested.

Because the entire control logic of an agent is auto generated out of model information, the verification process can be started on a more abstract level. Herewith the verification focuses on the identification of modeling failures in the previous planning phases and assumes individually implemented capability modules as pretested.

On this level of abstraction, function-oriented testing strategies are suitable to ensure the operability and capability of control components. A function-oriented test regards an agent as a black box. The black-box-testing is characterized by the fact that the test case is derived from the test specification (desired effect). Knowledge about the test object's internal structure is excluded from consideration.

In order to design test cases the specifications as well as the influencing variables of an agent need to be determined. Hereto information out of the planning phase of an agent will be reused.

The influencing variables can be derived from capability modules which cause changes in the agent's state variables in dependency of the control component's environment. For this purpose capabilities have to contain interfaces which can be categorized as following:

- interface to superior ERP systems in order to exchange job process information concerning manufacturing, handling or material processes
- interface to the object model in order to read and write certain object information on shop floor level
- interface to the actuator and sensor level in order to exchange information with device controllers.

With the usage of these three interfaces any desired system state of an agent can be created by setting a certain combination of job processes, object positions/relations and return values of

device controllers. These parameters represent the influencing variables. Messages will be considered as system internal values that can be affected by the three parameters.

The testing strategy comprises the following two goals:

1. only tested agents are allowed to be called by other agents
2. the interaction between control components should be in a scaleable manner

Thus, these two goals guarantee that the localization of the error cause can always be limited to a single agent and the error search-space does not stretch unnecessarily across several control components. Herewith, a high efficiency of the testing procedure can be achieved.

Based on these requirements an according testing systematic has to be developed which prevents from calling not yet completely verified control components during a test run.

B. Design Structure Matrix

To handle this problem the Design Structure Matrix (DSM) also known as Dependency Structure Matrix is being applied. The DSM provides a compact and clear representation of a complex system and a capture method for the interactions/interdependencies/interfaces between system elements.

This method is used to visualize complex relations in systems engineering or systems analysis, as well as in product-development and -planning. The matrix evaluates all elements of a system with respect to their dependency and particularly with regard to their dependency type. The outcome is used to derive all activities that are necessary to execute an activity as well as all the information that should be generated by an activity. By reading the columns of the matrix all elements that are being affected by an activity can be identified. Whereas reading the rows indicates which other elements the activity is depending on. Based on the DSM configuration the DSM algorithm generates an optimized execution sequence by selectively permuting columns and rows. This optimum is reached if all items of the matrix are arranged underneath the main diagonal, see Fig. 8 [7].

To apply this method for verifying a distributed, agent-based control system, the DSM is being used to analyse dependencies between control components. Hereto the agents as well as their process classes will be registered. The process classes represent the elementary functions within a FMS and can be classified in manufacturing, handling and transportation processes. Each

process class is assigned to an agent. Thereby, the job processing of an individual process class can require interactions between control components. In order to determine an optimal testing sequence the dependencies between control components have to be analyzed. Thereby the dependencies can be classified into two types:

- Interactions between two control components which can be avoided by certain combinations of influencing variables are referenced as *evitable interactions*
- Interactions between two control components which can not be avoided by certain combinations of influencing variables are referenced as *inevitable interactions*

Thus, two consequences can be drawn. Firstly, the influencing variables are suitable to define the scale of interaction. Secondly, certain interactions between control components can not be decoupled.

In order to receive an optimal testing sequence, the DSM algorithm is applied. As a result one can extract from the matrix *these control components which do not have any dependency followed by those with single or multiple dependencies..*

The ongoing example demonstrates the above presented approach on the base of the *cosmos 4 platform*. At first, the agents as well as the associated process classes are inserted into the matrix. According to the two derived dependency types (*evitable, inevitable*) the interconnection between agents are specified, see Fig. 8. In the following, the dependencies of the sawing machine KASTO will be illustrated in detail.

The dependency between the process class SAW_4WG of the agent KASTO and the process class PICK_PLACE of the agent ROB_GANTRY is determined as inevitable because KASTO can only finish its sawing job if the sawed workpiece is placed on a corresponding transport pallet by the robotic palletizer ROB_GANTRY.

An evitable interaction between two control components will be illustrated on the interconnection of the agents KASTO and AGV. The process class TRANSP_PAL of the AGV can be avoided by establishing a relation (ASSIGN) between a storage pallet and the transfer station of the sawing machine KASTO in the object model. Hereupon the capability module *request_pallet* of the agent KASTO does not request an AGV and hence the process class TRANSP_PAL is evitable.

This example points out two facts which have been introduced before. Firstly, if the calling of the agent AGV should be prevented, a corresponding test scenario by using the influencing variables has to be created. Secondly, the agent KASTO can not be decoupled from the agent ROB_GANTRY. Thus the agent ROB_GANTRY has to be tested beforehand.

C. Testing concept

The usage of the above presented DSM provides an optimal test sequence in which control components can be verified successively [7]. In order to build up a comprehensive testing tool, in the following a test concept will be presented in which the DSM has been integrated, see Fig. 9.

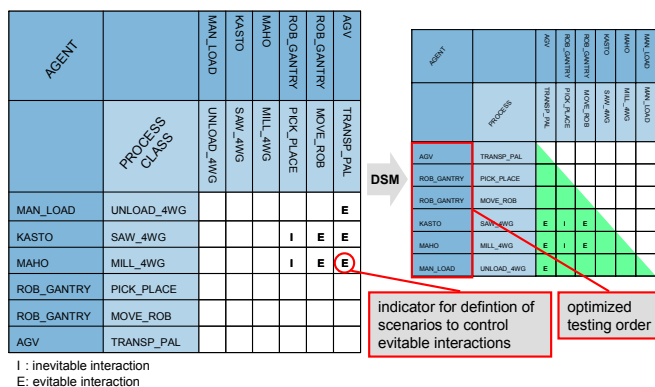


Fig. 8 Test case design based on the Design Structure Matrix

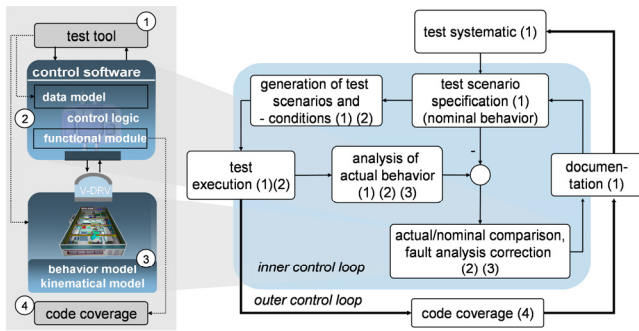


Fig. 9. Testing concept, outer and inner control loop

The concept consists of two control circuits, an inner and an outer control loop. The inner control loop is responsible for the test specification, execution as well as the analysis of the nominal and actual behavior of the system.

The test specification as well as the comparison of nominal and actual behavior is carried out manually by the user. Hereto information coming from the virtual kinematical model as well as the agent's user interface has to be taken into account. To document the operability and capability of the entire FMS control software, the test specification, as well as the test execution in the virtual kinematical model will be recorder and archived. Herewith the manufacturing equipment vendor is able to proof that the software is meeting its quality gates. In addition to the inner control loop the system will be completed by applying an outer control loop ensuring the test coverage. The outer control loop will be pictured in detail at the end of this paper.

D. Realization of a prototype

The test concept has been implemented as the *cosmos 4 verification* software tool and was integrated into the *cosmos 4e* engineering platform, see Fig. 10. The tool is divided into three components: *test configuration*, *test definition* and *test control*.

The *test configuration* allows the specification of dependencies between agents within the DSM as well as the creation of scenarios to control evitable interactions.

The *test definition* enables to specify test cases on the base of the *test configuration*. To define test cases, two perspectives are available. These are the *agent view* and the *product view*. The *agent view* focuses on the verification of a single control component and its associated process classes. Therefore, the control components are arranged in the optimized testing sequence. Based on a selected agent (e.g. KASTO), a list of supported process classes (e.g. SAW_4WG) will be displayed. Hereupon the corresponding parameters of the chosen process class will be shown and a test case in consideration of the influencing variables can be created.

If the selected process class has a dependency on another process class of the type "e=evitable", this one will be displayed. At this point the user is able to decide whether the process class and herewith a further agent should be included in or excluded from a test case. For this purpose predefined scenarios can be activated. The following example illustrates this connection in detail.

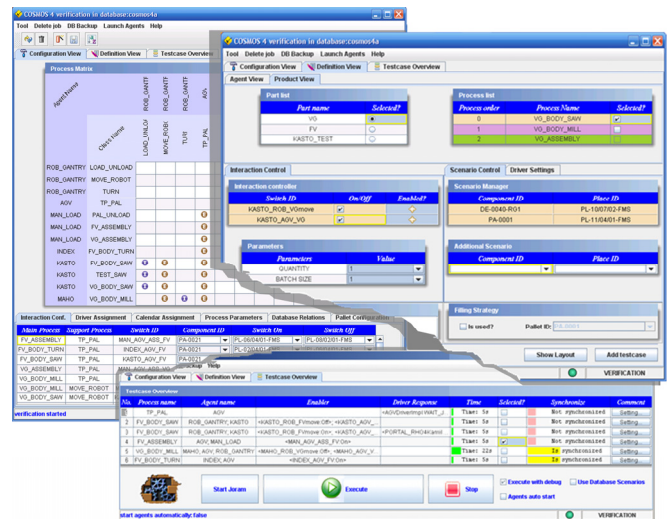


Fig. 10. *cosmos 4 verification* system

Assumed the user wants to verify the sawing process SAW_4WG. After choosing the desired process class the verification tool automatically detects on the base of the *test configuration* two evitable (MOVE_ROB; TRANSP_PLACE) and one inevitable (PICK_PLACE) processes. At this point the user can adjust the test range. The test case can comprise between two and four process classes.

In order to prevent all time consuming supply processes the user is interesting in downsizing the test range as much as possible during the first test run. Therefore the user activates only two of four processes. In the background the verification tool automatically generates two corresponding scenarios to prevent calling the processes MOVE_ROB as well as TRANSP_PLACE.

This general procedure can be applied to all participating agents of FMS control software. In case all control components are single tested, the user can switch to the *product view*.

The *product view* serves to verify the process chain of successive manufacturing operations in order to produce an entire product. Thereby the test can be scaled from a single operation up to the complete process chain. In a second step the several test cases for different products can be executed concurrently with increasing numbers. Herewith the presented test systematic ensures the step-wise ramp up from elementary operations up to maximum utilization.

After the test definition the coordination of the test execution is carried out. Here the test cases can be generated, managed and executed. This includes activating the control agents, the generation of the corresponding test scenarios in the data model as well as the synchronization with the virtual, kinematical shop floor model.

VI. VIRTUAL PROCESS QUALIFICATION

The virtual shop floor model allows the operation of the control system independently from the real shop floor. It provides the potential to analyze semantic relations between the control process definition and the behavior on the

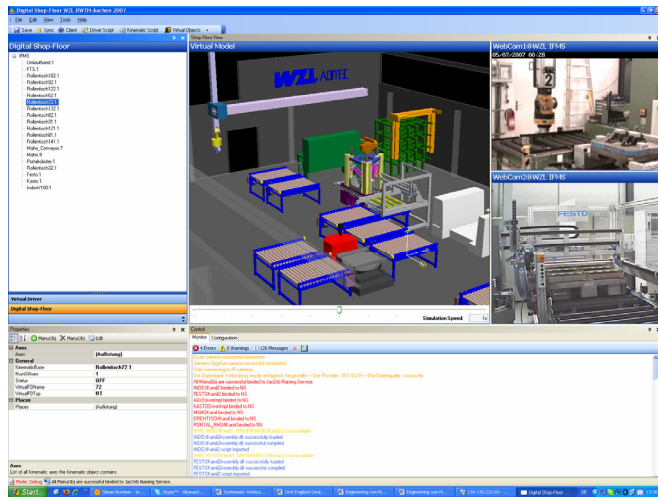


Fig. 11: Virtual process qualification with the Digital Shop Floor

actuator-sensor level. It also permits the identification of potential model errors. Therefore, the kinematical behavior of the manufacturing, handling and transportation processes is being emulated. The data flow between control software and shop floor model is carried out analogous to the real shop floor using so called virtual drivers that represent a service based interface. Further on, the answer pattern of the device controller (PLC) can be assigned in the virtual driver by the *cosmos 4 verification* tool in order to verify the operability of the corresponding function modules.

The concept for the virtual process qualification was implemented as a software tool. Based on the application programming interface (API) of the UGS Tecnomatix eM-RealNC software, a tool named “Digital Shop Floor” has been developed. Fig. 11 illustrates the graphical user interface of the software in whose centre the 3D visualization of the shop floor is located. The simulation speed is freely scalable and enables a faster test execution and thus accelerates the control software’s ramp-up significantly.

VII. CODE COVERAGE

The above presented method for systematic verification of flexible manufacturing control software can be ranked into the category of functional testing. Thereby, the system which needs to be tested can be considered as a black-box. An external perspective of the test object is used to derive test cases. Information about the degree of test coverage can not be gathered. For this, white-box testing techniques such as code-coverage will be applied [9]. Code-coverage is aware of the internal structure of the test system (control logic) and checks its internal execution paths. As a result, metrics (system of ratios) are created and provide a perception about the degree in which source code has been tested. On the basis of this finding, new tests can be generated.

An iterative application of both testing methods (systematic verification and code-coverage) provides the opportunity to maximize the quality of the entire control system.

Integrated into the *cosmos 4e* platform an open source code-coverage tool (EclEmma 1.2.2) to monitor the processing of the multi-agent system has been realized. illustrates an auto

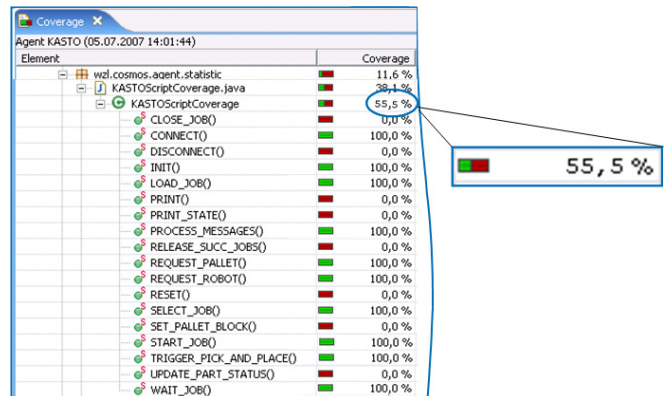


Fig. 12: Code-coverage report of the control logic of an agent (KASTO)

generated report about the code coverage of certain control logic parts of the agent KASTO. The coverage ratio represents 55,5% after the first verification run. Herewith, the user receives at any time feedback about the status of the verification phase and thus he is able to make a statement about the quality assurance based on these ratios.

VIII. CONCLUSION

The production ramp-up represents for the manufacturer as well as the user of automated production systems a large uncertainty and expense factor. In particular, the poor predictability of the increasing diversification of the product portfolio demands new approaches to ensure the early validation of operability and capability of automated production systems. Therefore, the developer has to cope with the dilemma of ensuring the operability and capability on the one hand and the need for a fast ramp-up on the other hand. The presented *cosmos 4 engineering* solution dissolves this goal conflict and represents a new approach to systematically verify a model based shop floor control system on the basis of a virtual process qualification.

REFERENCES

- [1] G. Lay; E. Schirmeister: *Sackgasse Hochautomatisierung?* Mitteilungen aus der Produktionsinnovationserhebung Nr. 22. Fraunhofer Institute for System Technology and Innovation Research, Karlsruhe, Germany 2001.
- [2] M. Weck: *Werkzeugmaschinen. Band 1 – Maschinenarten und Anwendungsbereiche.* 5. Auflage. Berlin: Springer-Verlag, 1997.
- [3] M. Weck: *Werkzeugmaschinen Band 4 – Automatisierung von Maschinen und Anlagen.* 6. Auflage. Berlin: Springer-Verlag, 2006.
- [4] C. Brecher, F. Possel-Dölken, and C. Almeida. *FMS control software with programmable control agents.* In CIRP Conference on Reconfigurable Manufacturing Systems, 2005.
- [5] Y. Cheng: *Entwicklung von Konzepten und Werkzeugen zur Modellbasierten Projektierung verteilter Fertigungsleitsysteme.* Diploma thesis. RWTH Aachen, 2006
- [6] R. Langen: *Methoden und Werkzeuge zur Erstellung von Fertigungsleitsoftware.* PhD thesis, RWTH Aachen, 1998.
- [7] A. Karlberger: *Entwicklung eines Werkzeugs zur systematischen Verifikation der Funktions- und Leistungsfähigkeit eines projektierten Fertigungsleitsystems.* Diploma thesis. RWTH Aachen 2007
- [8] C. Brecher, T. Buchner: *Modellbasiertes Engineering verteilter Fertigungsleitsysteme.* In VDI Fortschrittsbericht, Simulation in der Produktion, 1. Auflage VDI-Verlag 2006
- [9] J. Jenny Li, J. Robert Horgan. *χSuds-SDL: A Tool for Testing Software Architecture Specifications.* In Software Quality Journal, Volume 8, Number 4 / December 1999, pp. 241-253