Integrated Semantic Fault Analysis and Worker Support for Cyber-Physical Production Systems

Ingo Zinnikus, André Antakli, Patrick Kapahnke,
Matthias Klusch, Christopher Krauss, Andreas Nonnengart, Philipp Slusallek
German Research Center for Artificial Intelligence (DFKI)
Saarbrücken, Germany
E-Mail: FirstName.LastName@dfki.de

***Abstract*—About a decade ago, the fourth industrial revolution, also known as Industrie 4.0, has been ushered by the introduction of the Internet of Things and Services into the manufacturing environment. Since production and manufacturing control systems are increasingly networked and connected, the complexity of modern distributed cyber-physical production systems (CPPS) requires new tools for monitoring, failure detection and analysis. In this paper, we present a framework for semantic fault analysis of CPPS which combines semantic sensor data stream analysis for fault detection and diagnosis through reasoning on given domain model and belief network with fault prognosis through formal behavior analysis with timed hybrid automata. As CPPS are envisioned to not only cooperate with each other but also with humans on a new level of sociotechnical interaction, we use agent-based 3D visualisation tools to provide human users with support when repairing occurring faults. We illustrate the approach using the example of a smart factory case study.**

I. Introduction

About a decade ago, the fourth industrial revolution, also known as Industrie 4.0, has been ushered by the introduction of the Internet of Things and Services into the manufacturing environment. Industrie 4.0 is focused on creating smart products and processes flexibly in dynamic, real-time optimised and self-organising value chains, and profitably even down to production lot size of one. To rise up to this challenge, Industrie 4.0 applications basically operate on the principles and use of autonomous cyber-physical systems with self-* properties for integrated production across the entire value chain. In particular, the IP-networked and sensor-equipped machinery, systems, vehicles and devices of smart factories are vertically and horizontally integrated with service-based business processes both within a company and inter-company value networks. Together with the ever increasing general requirements of high flexibility, reduced delivery time, and short product life cycles, the Industrie 4.0 concept represents the highly dynamic, individualized, and networked environment of modern, digital factories. Besides, cyber-physical production systems are envisioned to not only cooperate with each other but also with humans on a new level of sociotechnical interaction.

There is a large number of challenges on the IT side for realizing Industrie 4.0: (i) The high flexibility of production processes requires the ability to quickly redesign and adapt production lines and all supporting processes in a company. (ii) The high variability of products with small batch sizes requires novel, highly adaptable ways to monitor the production line for quality and errors while providing support and training for workers that adapts to the current situation. (iii) To support quick changes we must move from fixed, specialized networks and interfaces to flexible architectures and service interfaces that can easily be reconfigured and support the low-latency, high-volume communication needed in industrial environments. This new approach to production is enabled by suitable IT tools that (partially) have yet to be developed: Industrie 4.0 requires the ability to model the production processes at a high level so that they can be adapted quickly and the ability to simulate, visualize, and verify these changes before execution [3]. Based on the simulated models we need the ability to derive monitoring strategies that ensure safe and high-quality production.

The individualised and self-organising, inter-connected production in CPPS breaks open the classical hierarchical control in production systems [18]. In order to prevent or reduce downtime, ensure reliability and maintain control in CPPS, the increasing complexity of modern networked and distributed production systems requires new tools for monitoring, failure detection and analysis. The human factor and support becomes even more important in manufacturing control as decisions and maintenance tasks have to be adapted to the complexity of the systems. We need the ability to derive support and training for those who work in these environments with new visualisation methods for humans in various roles, such as workers, supervisors, and managers.

To the best of our knowledge, the INVERSIV platform is the first 3D simulation platform for CPPS that flexibly integrates agent-based computing, semantic technologies, formal verification and 3D visualisation for production control. Besides, it is the first approach that combines semantic technologies and formal model-based verification for intelligent condition monitoring of machines.

The remainder of this paper is structured as follows. In section II we introduce the approach for fault detection and diagnosis combining model-based and semantic data analysis. We give an overview of the architecture, comprising the components for fault detection, agent-based worker visualisation and data synchronisation. In section III we describe in more detail the component for semantic data analysis, the approach for model-based fault detection using hybrid automata and

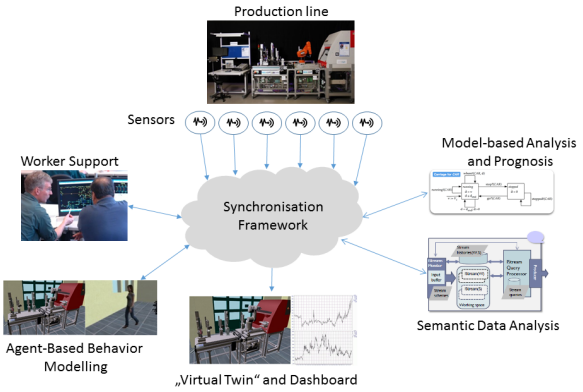

Fig. 1. Overview of the INVERSIV infrastructure.

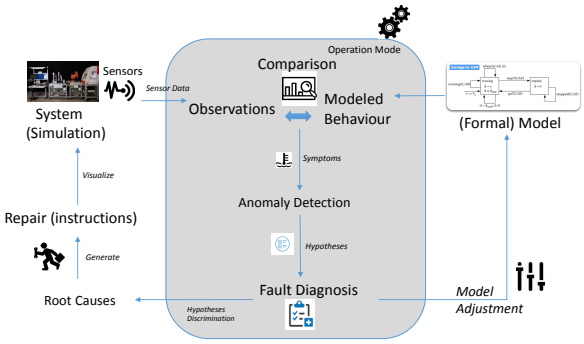

Fig. 2. Combining hybrid automata and semantic data analysis for model-based diagnosis.

the agent component for describing and visualising worker behavior. We apply the approach to a smart factory use case in section IV. We discuss related work in section V and conclude in section VI.

II. INVERSIV Platform Architecture

We assume a plant or production line equipped with multiple sensors which provide real-time data about the relevant modules (cf. fig. 1). The sensor data is gathered and streamed into the synchronization framework which provides data subscription endpoints in real-time for various clients. The sensor data is used to establish a dashboard and 'virtual twin' of the factory line which visualize the current state and processes running in the production line. For fault detection and prognosis, the model-based analysis component analyzes incoming data and communicates abnormal situations to the semantic data analysis component which diagnoses potential causes. Depending on the diagnosed cause, the agent-based modeling and execution environment is then used to visualise e.g. repair actions for operators and workers depending on their role in the production process.

Approaches to fault detection can be distinguished into phenomenological and model-based approaches (cf. [21]). In the phenomenological approach, using a trained classifier the sensor data is directly classified as correct or defective. In order to detect anomalies in the behavior in a model-based approach, a model is used to predict the normal – continuous and discrete – behavior of a plant. If the actual behavior – based on the observations – deviates from the nominal behavior, the observed behavior is classified as anomalous.

Both approaches have advantages as well as drawbacks: the phenomenological approach does not depend on a model but can detect failures only against the direction of causality. In the model-based approach which requires less data the relation between normal behavior and failure is much more comprehensible in case of a deviation but the problem domain has to be modeled manually (although there are attempts to learn these models, cf. [17]). The approach taken in this paper is to combine a model-based approach using hybrid automata and semantic data analysis in order to leverage the strengths of both approaches. The usage of hybrid automata allows at the same time a prediction about deviations occurring in the future.

In fig. 2, the combination of the approaches as developed in the project is represented in more detail. Based on sensor data which provides observations about a production line, model-based analysis detects deviations of observations from the model. For fault identification, potential root causes for the symptoms are then analysed, hypotheses generated and provided to the human user indicating the probability of the detected cause. At the same time, the formal model allows a prognosis of the evolution of the system based on the current data. The anomalous state(s) predicted for the future can again be analysed and root causes detected. Based on the analysis of the predicted failure, maintenance and repair actions can be generated which anticipate and prevent the failure before it may occur. Stable modifications and adjustments of the system behavior may be propagated into the formal model which then reflects the reconfiguration of the production line. Repair actions for the detected problem are generated and visualised to the human worker, using agent-based behavior modeling and 3D avatar visualisation.

In the following, we describe the corresponding components in detail and illustrate their interaction with reference to a smart factory use case.

III. Components

A. Semantic Stream Data Analysis with SDA

In agent-based simulations of productions with the INVERSIV platform, intelligent agents are supposed to continously monitor, detect and diagnose faults and conditions of the simulated machinery. For this purpose, we developed the INVERSIV platform component SDA for semantic sensor data stream analysis. The SDA combines techniques for semantic

stream reasoning with probabilistic reasoning in order to detect most likely symptoms, faults and conditions of monitored machinery, and to answer specific diagnosis queries. The semantic model of the considered SmartFactory in this project consists of a domain ontology and a domain belief network. The ontology defines the concepts and relations between the machines and their components, sensors, faults and symptoms in the description logic OWL2-DL, while the belief network represents the probabilistic knowledge on causal relations that are modeled in the ontology. The overall architecture of the SDA component is depicted in fig. 3.

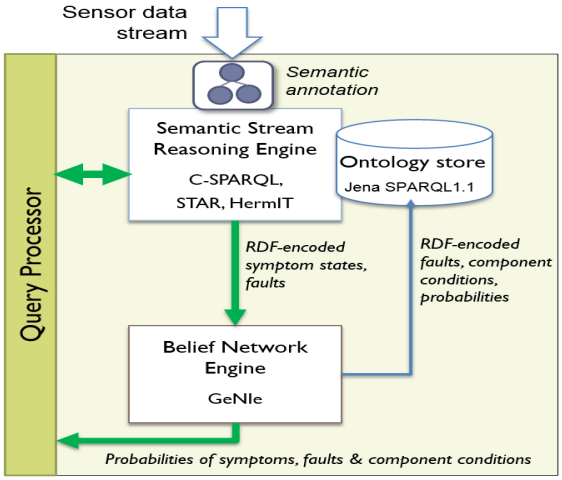

Fig. 3. SDA component architecture

The stream processing module of the SDA semantically annotates the simulated sensor data stream according to the domain ontology, and can continously answer given queries in C-SPARQL for fault symptom detection over the materialized stream data for each (consecutive) stream window. The SDA then performs probabilistic reasoning over the belief network in order to determine the evidentially most probable machine component fault for detected symptoms, as well as a list of probable faults and their symptoms for a detected machine condition as an initial diagnosis. Furthermore, SDA can perform semantic reasoning with its integrated reasoners STAR and HermiT either individually or in combination in order to answer a given set of semantic fault analysis (diagnosis) queries. Examples of such queries are: What is the most likely condition of given component, or Which other components are semantically affected by the detected fault, or what are the semantic relations between detected component faults [16]. The implemented SDA component utilizes the Jena triple-store (SPARQL 1.1), the C-SPARQL engine for RDF stream querying, the BN engine GeNIe for probabilistic reasoning, the reasoner STAR for RDF object-relational querying, the reasoner Hermit for reasoning in OWL2, and the OWL-API 3.4.3.

While the SDA component of the INVERSIV platform performs its analysis based on a semantic understanding of faults, their correlation with symptoms, their probabilities and their interrelations it does not consider any information about the potential behavior of the system. System models in HAVLE, however, exactly contain this information and in the context of failure analysis our goal is to leverage this knowledge to predict faulty situations. An example of the interplay between SDA and HAVLE for integrated detection, prognosis, and diagnosis of faults of a hydraulic machine in the smart factory scenario is provided in Sect. IV.

B. Machine Behavior Analysis with HAVLE

Analyzing potential behavior of such factory systems requires to consider both their continuous evolution, usually performed by the physical objects in the system, the discrete (usually control) actions and the interaction of both of those kinds of behavior. In general such systems are called *hybrid systems* and through the years *hybrid automata*, as introduced by Alur and Henzinger in [1] have established themselves as the methodology for the formal modeling and verification of hybrid systems.

Modeling and verification of hybrid systems happens at design time or at least independently to the running of the real system. The main idea in INVERSIV is to leverage the behavioral models and the description of desired or undesired behavior (given through properties) - that are both created as part of the analysis anyway - to monitor the behavior of the system at runtime. Such an approach allows the detection of inconsistencies between the actual and the modeled system, the detection of errors (i.e. propositional properties that do not hold in the current monitored state), and the prediction of potential undesired situations in the future based on the possible behavior of the system. The INVERSIV-System uses HAVLE[1] for modeling, verification and monitoring of hybrid systems.

1) Hybrid Automata: Hybrid automata (see figures 9 and 10 for examples) consist of a set of variables, locations, and transitions. Locations describe the modes the system can be in. In these locations the system evolves continuously. The continuous behavior is given through differential equations and constrained by invariants over the variables. Transitions are the discrete jumps between the different modes of the system. Jump constraints provide conditions on when a transition can be traversed and how variables change when it is. Additional conditions are imposed through parameterized multi labels, namely the existence of appropriate partners for synchronization. Finally, declaring a transition to be *urgent* results in the interruption of the continuous behavior in the source mode whenever the jump condition is met and potential partners for communication are present. Hybrid automata can be understood as finite, (in general) non-deterministic descriptions of all possible behaviors of a hybrid system (or components of a hybrid system).

2) Verification of Hybrid Automata: Based on rigorous formal semantics, given a model of the system, hybrid automata theory provides a formal understanding of the potential

[1]Hybrid Automata Verification by Location Elimination due to a verification algorithm based on location elimination [22] implemented in this tool.

behaviors of the system. Behaviors that should be allowed (or are desired) for the system are formulated as *properties* in a variation of temporal logic referring to the locations and variables of the system (see [22] for a formal definition). Properties considered in HAVLE deal in general with the question whether nothing bad will occur in all states that are reachable or - a dual view of this question - whether a desired situation can be reached. Verification itself is (as usual) performed on the states reachable from a set of initial states. HAVLE implements general reachability algorithms as known for example from [8] but also an elimination approach [22] especially suited for the purpose of monitoring.

Additionally to the mere answer to the question of whether some property holds or does not hold for the given system HAVLE also provides witness(es) for behavior(s) that violates the property. These witnesses are called traces and consist of interleaving descriptions of continuous evolutions of the system in modes and discrete steps between these modes.

3) Monitoring with Hybrid Automata: While in verification we have a well defined initial situation (the initial states of the automaton) and prove properties considering the states reachable from this initial situation, the essential observation when it comes to monitoring is that we have a new initial situation with every current state of the system. Also in verification we usually start by computing all states reachable from the initial states and then check whether all properties hold for those states. In monitoring we have the dual situation: we have one property (from which we can easily identify a set of immediate bad states, i.e. states in which the a-temporal part of the property already does not hold) and for which we can compute the set of states from which such immediate bad states are reachable. Strategies similar to backward reachability can be applied. This can be precomputed before runtime for every property to be monitored. At runtime for any current state of the system it only has to be checked whether this state is in this set for any property.

4) Architecture of the HAVLE *component:* Fig. 4 shows the different components and their relations. In a preparatory phase (before the actual runtime of the system) the hybrid automaton model is modeled in the HAVLE-Editor. Then the HAVLE-Verifier is configured to verify the properties to be monitored and to generate the required monitoring databases. Finally the translation mapping is defined that relates sensor data with hybrid automata model states. For the actual monitoring at runtime the HAVLE-Monitor is instantiated with the monitoring databases and this mapping.

The HAVLE-Monitor itself consists of two parts: a *sensor-state-translator* and a *matcher*. The sensor-state-translator receives a system state perceived through the current values of the available sensor data and translates it into a set of model states based on the mapping provided at initialization time. These model states will then be sent to a collection of state matchers - one for every property. Every matcher will - based on the provided database for the monitored property - check whether any of the states satisfies the monitored property. The matcher will finally return a monitoring result which contains

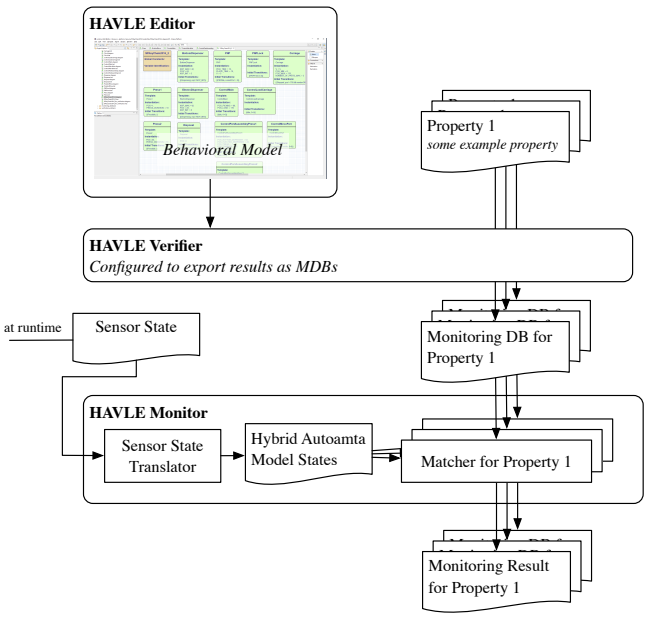

Fig. 4. HAVLE component architecture

the status of the check, i.e. whether the property holds or does not and, depending on the kind of the monitored property and its validity, a list of witnesses for this status. The provided witnesses consist of the location of the state together with a constraint on the variables that have been satisfied by the currently checked model state. Additionally, it contains a trace witness.

The interaction between the INVERSIV platform components HAVLE and SDA for fault detection, prognosis, and diagnosis in the smart factory scenario is exemplified in Sect. IV. In the virtual 3D simulation environment of the INVERSIV platform, workers represented as INVERSIV agents can use the result of such integrated semantic fault analysis of a machine to plan the respectively required maintenance action.

C. Agent-based Simulation of Workers with AJAN

The aim of our approach is to detect faults in cyber-physical production and manufacturing systems and to analyze possible reasons of these failures. The occurring faults have to be repaired by real workers working in such production settings. For this reason we support users of our INVERSIV system with 3D simulation tools, to visualize production faults and to give answers of how a human can repair them directly in the shop floor. For this, we need besides the production process also a representation of the worker and his behavior in the simulation tool. AJAN[2] is a component of the INVERSIV platform for simulating agent based workers in 3D environments.

To simulate human workers we need a system which provides the ability to simulate autonomous actors in the 3D visualization tool. In this context, the agent paradigm is often used as approach. The agent itself and his behavior is represented by

[2]AJAN is the acronym for Accessible Java Agent Nucleus and means agent in Turkish

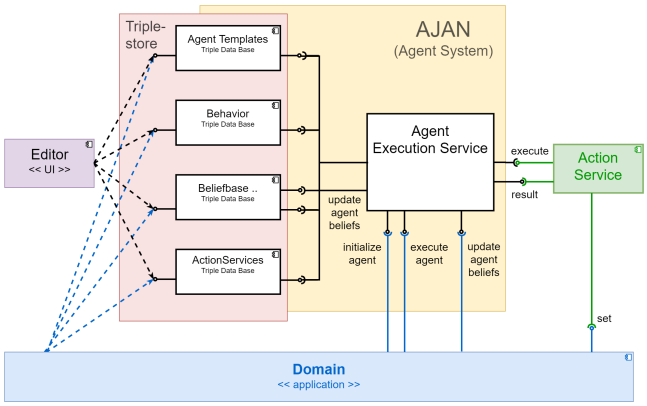

Fig. 5. Overview of the AJAN architecture.

an avatar and avatar movements in that 3D environment. To act autonomously in the simulation environment, the agent needs sensors and actuators to perceive other actors and to interact with them to accomplish its goals, like repairing production faults. In case of worker simulations, sensors can be realized for example by implementing a view frustum, for detecting 3D objects in the avatars field of view. With our approach of a combined formal model and a semantic sensor stream analysis to detect and analyze production failures, the agent can also use these upcoming information to react to the simulated production process. Using the sensor output and the available simulation actuators, the agent only needs a behavior model to execute contextually dependent and reasonable actions.

To perform a specific behavior by an agent, this behavior has to be modeled in advance. An established architecture to model agents is the Belief Desire Intention (BDI) [23] paradigm. Other agent languages, mostly used in the game industry, are the mathematical model Finite State Machine (FSM) [27] and its modular enhancement Hierarchical Finite State Machine (HFSM) [24], or the Behavior Tree (BT) [7], which was first mentioned in [11]. To model and execute agent behaviors with AJAN, we developed a SPARQL 1.1 extended BT architecture. As described before, one goal in INVERSIV is to support human users, especially those with less programming knowledge who work on site in manufacturing environments. BTs are developed to realize a modular agent architecture which is easy to extend, but which also provides an intuitive user interface to model different agent behaviors in a fast and simplified way.

AJAN is implemented as a web service which consists of four parts (cf. fig. 5): a knowledge base using a triple store; an execution service which receives environmental information and performs actions in the simulation tool; the sensors and actuators in the 3D environment; and a web editor to model BTs in a graphical way. In the triplestore the agent model, with the domain model and behavior model, as well as the instance knowledge are stored. The triplestore was implemented with the use of RDF4J[3] which provides SPARQL 1.1 endpoints to access the stored RDF graphs. The AJAN architecture also provides the use of other RDF stores like Fuseki[4]. The triplestore endpoints are used by the web editor, to store the agent models, and by the execution service, to load these agent models and to perform BTs. The execution service itself provides REST endpoints, to initialize and execute agents and to receive environmental changes for updating the agents knowledge base. To perform an agent, its behavior described in RDF is loaded by the execution service and translated into an executable BT. We implemented this JAVA based service with the RDF4J framework to handle RDF and the gdx.ai[5] library to run BTs. To perform actions in the simulation environment, e.g. by the execution service, these actions respectively the avatar animations are also implemented with REST endpoints. These endpoints expose a description[6] in an action language based manner as a RDF graph to a user in order (a) to know how to execute the action and (b) to get information about its environmental effects.

For the graphical web editor we extended the JavaScript Behavior3JS[7]-editor to model BTs in RDF and to define SPARQL queries for condition and action nodes. BTs can be built by drag and drop graphical elements and by connecting them with each other. These elements represent different kinds of BT-nodes, such as composite nodes, e.g. sequence or priority as well as parallel nodes, or decorator nodes, e.g. repeater nodes. *Composite nodes* decide the execution order of their child nodes. Instead, *decorator nodes* are used to decide how their child node has to be executed or how its output has to be propagated up to the tree. Decorator or composite nodes can have one or more child nodes which can also be decorator, composite or leaf nodes like action or condition nodes. *Leaf nodes* are using the agent knowledge to execute actions or to check if a defined condition holds. For accessing the RDF store, conditions or the input for actions are described with SPARQL queries. By using SPARQL 1.1 we are also able to model UPDATE[8] queries for manipulating the agents knowledge with update leaf nodes. Finally, the editor also provides an interface to load descriptions of environmental actions over their REST endpoints, to automatically generate action nodes which then can be used to model a BT.

IV. Use Case: Smart Factory Scenario

As use case for the INVERSIV platform, an example cyber-physical production system from SmartFactoryKL [32] with several networked and connected devices for key fob production has been selected. In particular, the considered smart factory (SF) module (see fig. 6) produces key fobs consisting of an upper and bottom shell and an intermediary

[3]A JAVA framework, formerly known as Sesame, is used in INVERSIV to create the agents belief base and to handle RDF: http://rdf4j.org

[4]The Jena SPARQL server: https://jena.apache.org/documentation/fuseki2/

[5]A JAVA based artificial intelligence framework. The feature we use is the Behavior Tree module: https://github.com/libgdx/gdx-ai

[6]The description of service actions respectively affordances is oriented to the action language A defined in [9]

[7]A Behavior Tree library written in JavaScript. For our editor to create BTs in RDF, we use the Behavior3JS-editor: http://behavior3js.guineashots.com/

[8]The W3C update language to manipulate RDF graphs: https://www.w3.org/TR/sparql11-update/

Fig. 6. Smart Factory production module in virtual 3D representation with dashboard containing sensor data and status information.

with an integrated USB stick for individualized data storage. This production module consists of several (hydraulically and electrically driven) presses for squeezing the parts together, a movable rotary arm (pick and place robot) which puts and removes the parts to and from the presses and a carriage moving on a conveyor belt for transportation of the parts. A number of dispensers provide additional material needed for the assembly process. The module is equipped with sensors which send status information about pressure, position and speed of the rotary arm and carriage, as well as the filling level of the various dispensers. The data is fed into the synchronization framework and production processes are visualized in real-time together with the relevant sensor data (cf. fig. 6).

A. Semantic Domain Model

The semantic domain model of this SF scenario represents the domain knowledge of an INVERSIV agent, and consists of two parts: A SF domain ontology (SF-Ont) in OWL2, and a SF belief network (SF-BN). In particular, the concept base of the formal SF-Ont ontology consists of about 500 concepts and relations, which define the semantics of the SF machinery, sensors, measured properties, component faults, symptoms, and conditions, as well as external factors and condition-fault-symptom relations (cf. fig. 7). The ontology part related to semantic fault detection and diagnosis is based on the vocabularies of the condition monitoring standards ISO 2041,13372,17359:2011, interviews with relevant domain experts, and an extension of the standard W3C SSN (semantic sensor network) ontology. The fact base of the SF ontology contains the descriptions of the concept instances, i.e. the concrete assets of the smart factory module in terms of its individual machines, their components and attached sensors, as well as RDF encoded data of sensor measurements, detected conditions, faults, symptoms and probability values as a result of the semantic sensor data stream analysis with the SDA component (cf. Sect.III-A).

The second part of the semantic domain model is the SF belief network, which models the probabilistic knowledge on causal relations that are defined in the SF-Ont ontology. In

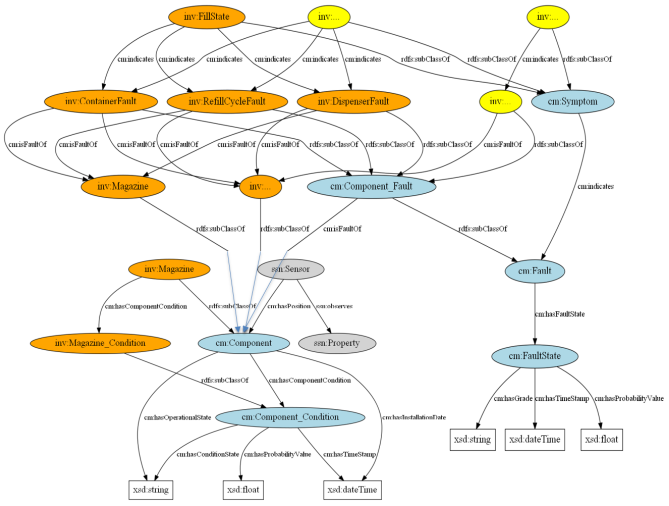

Fig. 7. Part of Smart Factory domain ontology

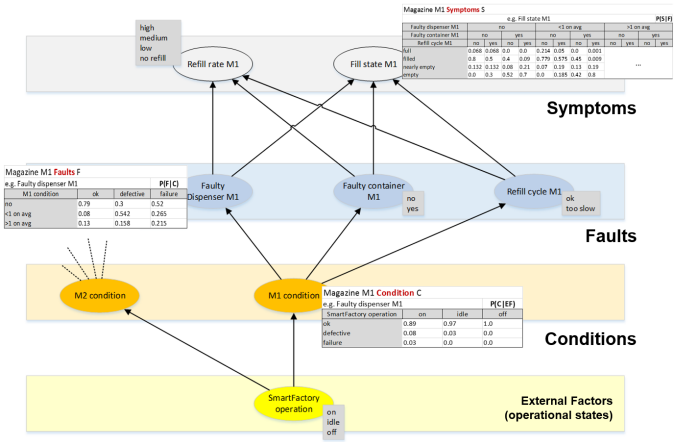

Fig. 8. Part of Smart Factory belief network for fault analysis

particular, the SF-BN (cf. fig 8) represents in a compact way the joint probabilities of cause-effect relations between the states of fault symptoms, faults, conditions, and external factors. The labels of nodes and their conditional probability tables in the SF-BN are the same as those for the respective concepts and relations that are defined in the SF-Ont. The conditional probability values are dynamically updated by the SDA after each fault symptom detection and used for probabilistic fault detection and diagnosis. For example, the SF belief network can be used to compute the most likely fault F based on detected fault symptoms S, i.e. the fault F with maximum conditional probability $P(F|S)$, and for the basic diagnosis of some condition C, i.e. set of symptoms S with $P(C|S) \geq \theta$.

B. Behavioral Model

For monitoring the behavior of the machinery in the scenario, a system model consisting of hybrid automata for the relevant physical objects, controllers and their interaction

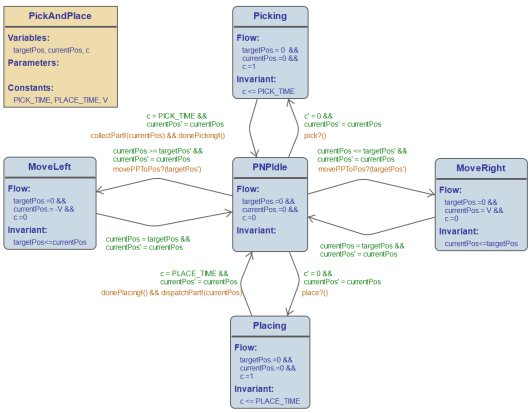

Fig. 9. Hybrid automaton model for Pick and Place Robot.

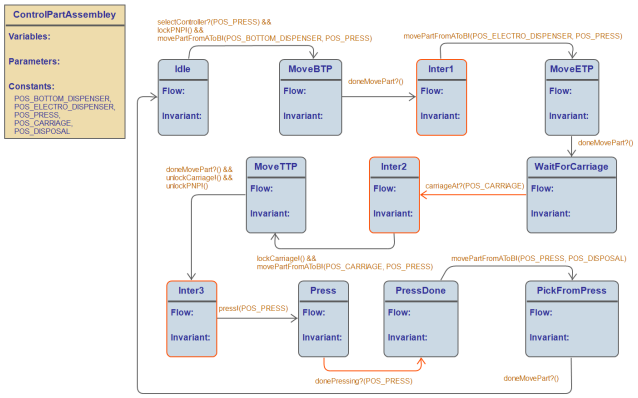

Fig. 10. Hybrid automaton model for Part Assembley Controller.

is modeled in the HAVLE-Editor. Figures 9 and 10 show hybrid automata for the *Pick and Place Robot* and the *Part Assembly Controller* as an example. Properties that describe the undesired situations that should be predicted or at least detected such as *'The magazine fill state of the bottom part dispenser should not become empty'* and *'The oil level in the accumulator of the hydraulic system should remain inside predefined bounds'* are formulated in terms of the formal model. Monitoring databases required for behavioral based monitoring in the HAVLE-Monitor are computed using the HAVLE-Verifier based on the system model and these properties. The sensor state mapping is formulated using data like the current position of the pick and place robot, the load of the press, and the current number of parts in the dispensers.

C. Integrated fault detection, prognosis, and diagnosis

Faults of the monitored hydraulic drive of the press in the SF scenario are online detected by SDA for its current states. As mentioned above, this detection is based on structured domain knowledge about concepts and causal relations between symptoms and faults, which is represented in standard description logic and a belief network. However, the static logic-based semantic and probabilistic knowledge representation and reasoning techniques used by SDA (cf. Sect. III-A) do not allow to determine whether and when some fault state of the hydraulic drive, thus the hydraulic press, could be reached from the current state in the future. The HAVLE component, on the other hand, is capable of such a prognosis as it has knowledge about the potential behavior of the system through the hybrid automaton model. Using the precomputed monitoring databases (one for every monitored property) it can check whether an undesired situation is reachable from the current state. However, unlike SDA, HAVLE does not have any formally specified notions of faults, symptoms and causal relations between them, hence cannot diagnose, that is determine the most probable causes of some faulty situation it predicts. The diagnosis of the predicted faulty situation has to be performed by SDA, again.

As an example of such an interplay in the smart factory scenario, suppose that HAVLE monitors the desired property 'SmartFactory produces 2 key chains per minute' over the sensor data, while SDA detects the fault symptom 'High pressure at press P2'. After SDA notifies HAVLE about this event, the latter re-verifies the property using an updated model. In particular, the automaton for the press is replaced by a different instance with a slower working cycle. The formal model used by HAVLE is thus updated based on currently observed runtime behavior by SDA.

Now suppose that the HAVLE-Monitor detects that - considering the current configuration of the system - it is possible for the bottom part dispenser to become empty in some future state R in 4:38 min (violating a monitored property). The predefined mapping allows HAVLE to translate this property together with its validity into a fault symptom understandable by SDA, i.e. 'Magazine fill state empty' in R. The trace provided together with this prediction contains the behavior required to get from the current state to the actual undesired (faulty) state R. Though SDA detected no fault symptom for the current state of the whole machinery, it is now notified by HAVLE about this prognosticated type of fault. SDA extracts the evidence for the fault symptom 'Magazine fill state empty' from the trace by use of the predefined mapping to concepts in the SF domain ontology, and then uses its SF belief network to diagnose the faulty state R of the magazine.

In particular, the result of the probabilistic reasoning by SDA is returned in textual form as 'Magazine refill cycle too long [prob 0.72]' and 'Magazine in bad condition [prob 0.79]' to INVERSIV agents, or INVERSIV dashboard user. In other words, SDA extracts the type S and state X. For determining the related most likely fault type, it then feeds this evidence and the actual results of C-SPARQL stream queries for all other fault symptoms as additional evidences into its domain belief network. The detection returns the fault type F and its state s with maximum conditional probability given these evidences, i.e. F with maximal $P(F = s|S = X...S_n = X_m))$, and the most likely condition type C with state c of the monitored machine or component such as the magazine in the example above. This corresponds with the intuitive notion of diagnosis of the faulty state returned by HAVLE (fault and condition which caused the observed symptoms to appear). As mentioned above, in condition monitoring, basic

diagnosis refers to the determination (filtering) of the most likely (observed) symptoms S that are caused by the detected fault/condition states (S with $P(C = c|S = X) \geq \theta$).

The SDA component can also perform other kinds of semantic reasoning over the SF-Ont for fault diagnosis. As an example, suppose that SDA detects low cooling power in the hydraulic press of the smart factory module. This symptom is detected as a result of the evaluation of the respective C-SPARQL symptom query over the stream data, that is the checking of the temperature difference between entry and outlet of the cooling unit of the hydraulic drive of the press. In this example, SDA uses the SF-BN to determine that this low cooling power is most likely caused by the onset of a certain fault, that is a cooling circuit leakage, with highest probability of .84. According to interviewed domain experts, one interesting semantic fault diagnosis is to determine which other components of the press could be affected by this detected fault, and which sensors are involved. This diagnosis query is answered by SDA through query answering with SPARQL and reasoning on symptoms-faults-components relations in the SF-Ont ontology with two different semantic reasoners STAR and HermIT. The STAR reasoner computes the shortest paths between given instances in the ontology, and the HermIT reasoner is used for logical concept classification into the ontology.

In particular, SDA uses the STAR reasoner to check for each component X of the hydraulic drive of the press, whether there is a shortest semantic relation path in the SF-Ont ontology between X and the fault instance Y (cooling circuit leakage) via symptoms S1 of Y and S2 of X with some shared property P. In our example, the control valve and the accumulator are detected as being possibly affected by the cooling circuit leakage according to their semantic relations modeled in the domain ontology. For each inferred property-symptom relation (P, S1, S2), SDA then uses the reasoner HermIT to identify sensors z with which the shared property P of the symptoms s1, s2 can be observed through classifying the respectively abstract sensor concept $QC \equiv Sensor \sqcap \exists observes.(P \sqcap \exists monitorsSymptom(S1 \sqcap S2))$ into the SF-Ont ontology and retrieve the relevant sensor concept instances. Finally, SDA uses the BN tool GeNIe to determine for both affected components, the valve and the gas accumulator of the hydraulic drive of the press, the most probable condition state given the detected fault state. The overall diagnosis result is compiled into a given query-specific explanation form (textual or tabular) that can be shown to a human worker by an INVERSIV agent.

D. Agent-based worker support

In response to the returned report on the detected fault of the hydraulic press, the human worker as user of the INVERSIV dashboard has the option (a) to plan and execute the required maintenance action directly, or (b) activate the INVERSIV agent as representative of the human worker to come up with such a repair plan and visualize it for him as operative instructions in the real-world smart factory environment. In the latter case, the INVERSIV agent adapts BTs predefined for different fault types to the concrete fault situation, and generates the appropriate 3D visualisation of executing the repair action plan. As mentioned above, the INVERSIV component AJAN allows to configure an INVERSIV agent in support of its (repair) action planning at any time.

Fig. 11. 3D Visualisation of repair action for worker

In particular, AJAN enables the system configurator to create different kinds of agents which control avatars in the simulation environment to test how a worker should behave in critical situations, and how this behavior influences the subsequent production. With the AJAN-editor all possible avatar animations – respectively actions – implemented in the INVERSIV simulation environment, like movement animations, navigation or production flow manipulations, are accessible to model agent behaviors with a graphical programming language. Because of the modularity of the BT paradigm, there is no need in AJAN to repeatedly model the same behavior for different contexts. Instead, predefined BTs can be used to create new kinds of behaviors. By using a graph based knowledge representation, an AJAN agent is adaptable for different kinds of simulation scenarios without redesigning database schemata or any downtime of the AJAN web service. With SPARQL 1.1, the agent developer can use a well known, flexible and extensive query language to work with agent beliefs for graph checking and updating.

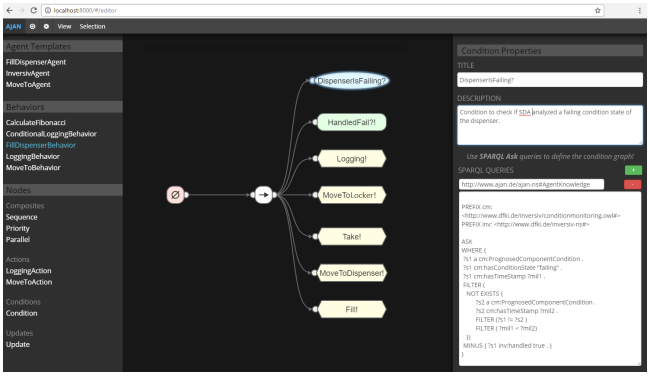

Fig. 12. Worker behavior modeled in the AJAN BT-Editor

In our SF use case scenario, a possible fault is an empty

dispenser which leads to a production stop. Therefore, the worker has to replenish a dispenser before it runs empty. The worker must be informed about a critical filling state, so that he has enough time to refill the dispenser for a smooth production. To visualize this specific situation, we modeled an AJAN agent to simulate a worker who refills the critical dispenser. We assume that the needed parts to refill this dispenser are stored in a locker. For this reason, the worker must first go to that locker, take the parts and walk to the dispenser before he can refill it. In fig. 12 the agent behavior with the sequence of these worker actions (yellow nodes) is shown. This sequence (arrow labeled node) is executed from top to bottom. By performing BT actions, avatar movements are triggered in the visualization tool and the filling state of the dispenser will be manipulated. As mentioned before, the agent needs information about that critical filling state to behave at a reasonable time. This INVERSIV agent receives detected failing results (in OWL2) from the SDA component and stores these in its belief base. To use this knowledge to decide if the worker has to refill the dispenser, a SPARQL query is used to check this incoming critical situation with a condition (blue node). Fig.11 shows the 3D visualization of the agent controlled worker of our SF scenario, in this case while refilling the critical dispenser. After running the simulation, the dashboard user can see if the simulated worker behavior was ergonomically and for the production flow optimal. Furthermore the real worker gets an instruction which actions have to be done in order to refill the dispenser.

V. Related Work

The interplay between the INVERSIV components SDA and HAVLE for integrated fault detection, prognosis, and diagnosis is related to work on intelligent condition monitoring. For example, in [10], intelligent fluid condition monitoring of wind turbine gears is performed through semantic sensor data analysis offline by applying semantic technologies for interpreting the state of turbine parts and answering questions related to their maintenance. Similar to the work in [16], [12], the specific domain knowledge is encoded in OWL2 and with SPIN rules; given fault detection and diagnosis queries are answered by use of the semantic reasoners Fact++, STAR, TopSPIN rule engine over a central SwiftOWLIM store. In [16], an intelligent condition monitoring system for hydraulic drives has been developed that combines statistical, probabilistic and semantic data analysis for fault detection and diagnosis with semantic explanations to the user. The semantic analysis component exploits RDF stream processing with C-SPARQL, semantic query answering offline with SPARQL, and semantic reasoning online with STAR and HermiT either individually or in combination in order to answer a given set of fault analysis queries as required and with reasonable response times. In fact, our work is largely inspired by this system.

There are quite a few approaches on fault detection, diagnosis and prediction based on behavioral models of hybrid systems (for a general overview cf. [4]). [15] base their prognosis on an extension of timed automata. [30] compares measured event time sequences with a stochastic timed discrete event model given as stochastic timed automaton. In [25] the authors present a framework for complex systems that combines diagnosis and prognosis. [26] also considers diagnosis and prognosis based on state representations that provide the nominal behavior of the system and the progression of faults. By that they are able to predict the remaining useful life (RUL). Similarly [6] tries to provide predictions on the RUL based on extensions of hybrid automata with fault progression and degradation functions. Integrating model-based and statistical methods in the diagnostic scheme [29] presents an approach for detecting and diagnosing faults in hybrid systems. [19] perform model-based diagnosis of hybrid systems using hybrid bond graphs. In [17] the authors concentrate on the automated generation of behavioral models as timed automata from a system running in nominal mode. Then anomaly detection in the actual system is performed against this model. However, the techniques we use for the detection and prediction of undesired situations are very different. Our models are not explicitly created for the purpose of monitoring. They provide a description of the potential system behaviors including such that lead to undesired situations. These are specified through properties rather than by modeling faults into the system model. We thereby distinguish between system behavoir and desired properties of the system. Also our approach for the prognosis is based on the application of verification techniques.

Several commercial systems allowing the configuration and 3D simulation of production environments in the shop-floor context, e.g. *Tecnomatix*[9], *FlexSim*[10], *visTABLEtouch*[11] or *SIMUL8*[12]. *DELMIA*[13] for example, is a tool which allows additionally the validation of 'produced' products and the evaluation of manufacturing processes. *DELMIA* is also used in [14] for prototyping CPPS environments. To our knowledge, none of these solutions dealing with internal machine models, let alone the fault detection and diagnosis.

In the CPPS context, multi-agent technologies are mostly used for production control and planning, e.g. in [28], in [20] or in [5]. *AnyLogic*[14] instead, that advertises its use for production process visualization, uses FSM agents to simulate worker behaviors. Other 3D simulation platforms using agent based technologies to simulate workers are presented in [13] and [31]. In [13] BDI agents are coupled with planning techniques and in [31], the main focus is the collaborative configuration and validation – using hybrid automata – of 3D factories. In [2], a three layered approach is presented for simulating 'intelligent' worker behavior in 3D, in which the decision making process of BDI-agents and the simulation domain semantics is modeled in Answer Set Programming (ASP). Most of these solutions offer no graphical user interface

[9]Tecnomatix: plm.automation.siemens.com/Tecnomatix
[10]FlexSim: www.FlexSim.com/FlexSim
[11]visTABLEtouch: www.vistable.de/visTABLEtouch-software
[12]SIMUL8: www.SIMUL8.com
[13]DELMIA: www.transcat-plm.com/software/ds-software/delmia
[14]AnyLogic: www.anylogic.com

for an intuitive and clear agent modeling or are unnecessarily complicated for modeling complex behaviors.

However, there is no implemented approach of combining semantic technologies and verification with hybrid automata for intelligent condition monitoring as done in the INVERSIV platform.

VI. Conclusion

In this paper we presented a novel framework for semantic fault detection in CPPS combining model-based analysis and semantic data analysis. Sensor data from a production line is fed into a synchronization middleware which distributes the data among the connected components. The sensor data and the state of the production module is visualised in real-time in a dashboard and a 3D 'virtual twin'. The model-based analysis component uses hybrid automata to detect and predict undesirable states. The semantic data analysis component then generates hypotheses and potential causes for the predicted behavior. An agent-based modeling and execution component is used to visualise repair actions for workers and operators, depending on the diagnosed cause.

Acknowledgment

The work described in this paper has been funded by the German Federal Ministry of Education and Research (BMBF) through the project INVERSIV under grant 01IW14004.

References

[1] Alur, R.; Courcoubetis, C.; Henzinger, T. A.; Ho, P.-H. (1993): Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems. Hybrid Systems. Springer, Lecture Notes in Computer Science.

[2] Antakli, A.; Zinnikus, I.; Klusch, M. (2016): ASP-Driven BDI-Planning Agents in Virtual 3D Environments. Proceedings of 14th German Conference on Multiagent System Technologies (MATES), Klagenfurt, Austria; LNAI 9872, Springer.

[3] Bauernhansl, T.; ten Hompel, M.; Vogel-Heuser, B. (2014): *Industrie 4.0 in Produktion, Automatisierung und Logistik - Anwendung, Technologien, Migration*. Springer.

[4] Blanke, M.; Kinnaert, M.; Lunze, J.; Staroswiecki, M.; Schröder, J. (2006): *Diagnosis and Fault-Tolerant Control*. Springer.

[5] Block, C.; Morlock, F.; Dorka, T.;Kuhlenktter, B. (2016): A Human Centered Multi-Agent-System for Production Planning and Control. In Applied Mechanics and Materials (Vol. 840, pp. 132-139). Trans Tech Publications.

[6] Chanthery, E.; Ribot, P. (2013): An Integrated Framework for Diagnosis and Prognosis of Hybrid Systems. In: Proceedings Third International Workshop on Hybrid Autonomous Systems.

[7] Dawe, M.; Gargolinski, S.; Dicken, L.; Humphries, T.; Mark, D. (2014): Behavior Selection Algorithms: An Overview. Game AI Pro - Collected Wisdom of Game AI Professionals, Ed.: S. Rabin, pp. 47-60, CRC Press.

[8] Frehse, G. (2005): PHAVer: Algorithmic Verification of Hybrid Systems Past HyTech. Hybrid Systems: Computation and Control, 8th International Workshop. Springer, Lecture Notes in Computer Science.

[9] Gelfond, M.; Lifschitz, V. (1998): Action languages. In: Electronic Transactions on AI. Vol. 3.

[10] Guenel, A.; Meshram, A.; Bley, T.; Klusch, M. Schuetze, A. (2013): Statistical and Semantic Multisensor Data Evaluation for Fluid Condition Monitoring in Wind Turbines. Proceedings of 16th International Conference on Sensors and Measurement Technology (SENSOR).

[11] Isla D. (2005): Handling Complexity in the Halo 2 AI. http://www.naimadgames.com/publications/gdc05/gdc05.doc

[12] Jin, G.; Xiang, Z.; Lv, F. (2009): Semantic Integrated Condition Monitoring and Maintenance of Complex System. Proceedings of 16th International Conference on Industrial Engineering and Engineering Management.

[13] Kapahnke, P.; Liedtke, P.; Nesbigall,S.; Warwas, S.; Klusch, M. (2010): ISReal: An Open Platform for Semantic-Based 3D Simulations in the 3D Internet. Proceedings of 9th International Semantic Web Conference (ISWC), LNCS 6414, Springer.

[14] Kashevnik, A.; Teslya, N.; Yablochnikov, E.; Arckhipov, V.; Kipriianov, K. (2016): Development of a prototype Cyber Physical Production System with help of Smart-M3. In Industrial Electronics Society, IECON 2016-42nd Annual Conference of the IEEE (pp. 4890-4895). IEEE.

[15] Khoumsi, A. (2009): Fault Prognosis in Real-Time Discrete Event Systems. 20th International Workshop on Principles of Diagnosis.

[16] Klusch, M.; Meshram, A.; Schuetze, A.; Helwig, N. (2015): ICM-Hydraulic: Semantics-Empowered Condition Monitoring of Hydraulic Machines. Proceedings of 11th International Conference on Semantic Systems (SEMANTiCS); Vienna, Austria; ACM

[17] Maier, A.; Niggemann, O.;Vodencarevic, A.; Just, R.; Jaeger, M. (2011): Anomaly Detection in Production Plants Using Timed Automata. Proceedings of 8th International Conference on Informatics in Control, Automation and Robotics (ICINCO). Noordwijkerhout, The Netherlands.

[18] Monostori (2014): Cyber-physical production systems: Roots, Expectations and R&D Challenges. *Procedia CIRP*, 17:9-13.

[19] Narasimhan, S.; Biswas, G. (2007): Model-Based Diagnosis of Hybrid Systems. In: IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans IEEE.

[20] Nishioka, Y. (2004): Collaborative agents for production planning and scheduling (CAPPS): a challenge to develop a new software system architecture for manufacturing management in Japan. International Journal of Production Research, 42(17), 3355-3368.

[21] Niggemann O.; Lohweg V. (2015): On the Diagnosis of Cyber-Physical Production Systems: State-of-the-Art and Research Agenda. Proceedings of 29th AAAI Conference on Artificial Intelligence, Austin, Texas.

[22] Nonnengart, A. (2000): Hybrid Systems Verification by Location Elimination. Proceedings of the 3rd International Workshop HSCC 2000 Springer, Lecture Notes in Computer Science.

[23] Rao; Anand, S.; Georgeff, M. P. (1995): BDI agents: From theory to practice. Proceedings of International Joint Conference on Multi-Agent Systems (ICMAS). AAAI, pp. 312-319.

[24] Risler, M. (2010): Behavior Control for Single and Multiple Autonomous Agents Based on Hierarchical Finite State Machines. Dissertation, TU Darmstadt, Germany, tuprints.ulb.tu-darmstadt.de/2046

[25] Ribot, R.; Pencole, Y.; Combacau M. (2009): Diagnosis and Prognosis for the Maintenance of Complex Systems. Proceedings of IEEE International Conference on Systems, Man, and Cybernetics, USA, doi:10.1109/ICSMC.2009.5346718.

[26] Roychoudhury, I.; Daigle, M. (2011): An Integrated Model-Based Diagnostic and Prognostic Framework. In: Proceedings of 22nd International Workshop on Principles of Diagnosis, Murnau, Germany.

[27] Wagner, F.; Schmuki, R.; Wagner, T.; Wolstenholme, P. (2006): *Modeling Software with Finite State Machines: A Practical Approach*. CRC Press.

[28] Vogel-Heuser, B.; Diedrich, C.; Pantförder, D.; Göhner, P. (2014): Coupling heterogeneous production systems by a multi-agent based cyber-physical production system. In Industrial Informatics (INDIN), 2014 12th IEEE International Conference on (pp. 713-719). IEEE.

[29] Zhao, F.; Koutsoukos, X.; Haussecker, H.; Reich, J.; Cheung, P. (2005): Monitoring and fault diagnosis of hybrid systems. In: IEEE Trans Syst Man Cybern B Cybern. IEEE.

[30] Zemouri, R.; Faure, J. M. (2006) Diagnosis of discrete event system by stochastic timed automata. IEEE Conference on Computer Aided Control System Design. IEEE.

[31] Zinnikus, I.; Spieldenner, T.; Cao, X.; Klusch, M.; Krauss, C.; Nonnengart, A.; Slusallek, P. (2013): A Collaborative Virtual Workspace for Factory Configuration and Evaluation. Proceddings of IEEE 9th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom); Austin TX, USA; IEEE Press.

[32] Zuehlke, D. (2010): SmartFactory Towards a factory-of-things. *Annual Reviews in Control*, 34(1):129138, Elsevier.