Supporting Runtime Decision Making in the Production Automation Domain Using Design Time Engineering Knowledge

Thomas Moser[1], Wikan Danar Sunindyo[1], Munir Merdan[2] and Stefan Biffl[1]

[1] Christian Doppler Laboratory
"Software Engineering Integration for Flexible Automation Systems"
Vienna University of Technology, Vienna, Austria
[2] Automation Control Institute
Vienna University of Technology, Vienna, Austria
{thomas.moser, wikan.sunindyo, munir.merdan, stefan.biffl}@tuwien.ac.at

Abstract. Complex production automation systems are often represented as multi-agent systems which need to be reconfigured correctly and efficiently to adapt to new requirements. While the system knowledge is available at design time in form of workshop layouts, product trees or production strategies, at runtime this knowledge is often not available and therefore not used at all for operational decision making. In this paper we describe an engineering ontology used for the representation of design time engineering knowledge for supporting runtime decisions. We evaluate the proposed approach using three scenarios from the production automation domain. Major result was that the explicitly available design time knowledge can provide valuable input to runtime decisions.

Keywords: Runtime decision making; design time knowledge, engineering ontology.

1 Introduction

Complex software-intensive systems in production automation need to be flexible to adapt to changing business situations and to become more robust against relevant classes of failures. Production automation systems consist of components, for which a general design and behavior is defined during the design phase, but much of the specific design and behavior is defined during implementation, deployment, and run time with a range of configuration options. The "Simulator for Assembly Workshops" (SAW) [1] simulates complex reconfigurable production automation systems to maximize the overall system output by scheduling sequences of transport and machine tasks over 100 times faster than the actual hardware at VUT's ACIN lab[1]. Figure 1 (left hand side) illustrates an example assembly workshop layout that consists of software-controlled manufacturing components: transport components such as conveyor belts (dark green), crossings (light green), and stoppers (yellow/red circles); and assembly machines (colored rectangles with round corners); product parts are trans-

[1] Automation & Control Institute; http://www.acin.tuwien.ac.at

ported on pallets (colored rectangles; colors represent the target machines). SAW has been validated with real hardware components in an assembly workshop lab to ensure that the simulation outcome is relevant for real production automation systems.

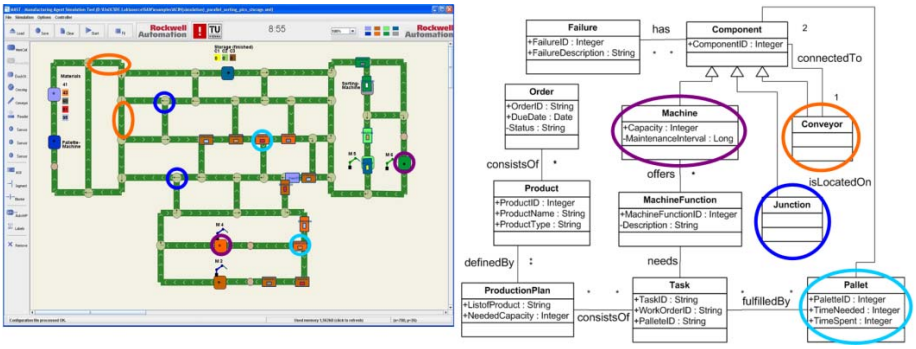

Figure 1. SAW Simulator and underlying data model.

Engineers, who want to adapt the system at runtime, need information from software models that reflect dependencies between components at design and run time, e.g., the workshop layout, customer orders and assembly procedures that translate into needs for machine function capacities over time; and the coordination of tasks for redundant machines in case of a failure. During development, design-time software models like data-oriented models (e.g., class or EER diagrams) or workflow-oriented models (e.g., sequence diagrams or state charts) are the basis to derive run-time models. But these models are often not provided in machine-understandable format to reflect on changes at runtime, i.e., the knowledge is modeled using an explicit human-understandable way but cannot be accessed by components automatically. Domain and software experts are needed to integrate the fragmented views (e.g., propagating model changes into other models, cross-model consistency checks) from these models, which often is an expensive and error-prone task due to undetected model inconsistencies or lost experience from personnel turnover.

Practitioners, especially designers and quality assurance (QA) personnel, want to make reconfigurable software-intensive systems (which like SAW consist of components defined by general design-time behavior, derived run-time configuration, and run-time specific behavior enactment) more robust against important classes of failures: machine failures, misuse from invalid supply, and failure-related changes in machine capacities at runtime. QA people could benefit from more effective and efficient tool support to check system correctness, by improving the visibility of the system defect symptoms (e.g., exceptions raised from assertions).

Challenges to detect and locate defects at run-time originate from the different focus points of models: e.g., components and their behavior are defined at design time, while configurations may change at runtime and violate tacit engineering assumptions defined in the design-time models. Without an integrated view on relevant parts of both design time and runtime models inconsistencies from changes and their impact are harder to evaluate and resolve between design and run time. Better integrated engineering knowledge can improve the quality of decisions for run-time changes to

the system, e.g., better handling severe failures with predictable recovery procedures, lower level of avoidable downtime, or better visibility of risks before damage occurs.

In this paper we present an approach to improve the support for runtime decision making with an engineering ontology. This engineering ontology provides a better integrated view on relevant engineering knowledge in typical design time and runtime models, which were originally not designed for machine-understandable integration. The engineering ontology can contain schemes on all levels and instances, data, and allows reasoning to evaluate rules that involve information from several models that would be fragmented without machine-understandable integration. The major advantage of using the engineering ontology for representing and querying the domain-specific engineering knowledge is the fact that ontologies are well suited to model logical relationships between different variables in axioms which can be used later for the derivation of assertions based on measured runtime data. We illustrate and evaluate the engineering ontology approach with three exemplary scenarios (change of conveyor directions, machine reconfiguration and machine maintenance preparation) from the production automation domain. Major result was that the explicitly available design time knowledge can provide valuable input to runtime decisions.

The remainder of this paper is structured as follows: Section 2 summarizes related work on Production Automation Systems, on the representation of design time knowledge and on the support of runtime decisions. Section 3 identifies the research issues and introduces the use case. Section 4 presents the approach; and finally section 5 discusses the findings, concludes the paper and presents further work.

2 Related Work

This section summarizes related work on Production Automation Systems, and on the representation of design time knowledge and the support of runtime decisions.

2.1 Production Automation Systems

By offering modularity and decentralizing system control, multi-agent-based approaches are recognized as a promising way to reduce complexity and increase flexibility of manufacturing systems [2, 3]. In this context, an agent is an intelligent entity placed in a manufacturing environment in order to supervise particular units and make decisions that influence the environment as well as its state. Agents communicate and negotiate with each other in order to perform the operations based on the available local information or in order to solve possible conflicts. However, manufacturing systems typically consist of heterogonous units, which use different types of data and data structures, and it is not easy to ensure the uninterrupted flow of information between and sometimes through the controlled levels. In order to ensure the correct understanding of the exchanged messages, agents must have the same presentation of the environment, or at least that part of the shared environment about which they are exchanging information with each other. Ontologies have been developed and investigated for quite a while in artificial intelligence and natural language processing to facilitate knowledge sharing and reuse [4]. They are of vital importance for enabling

knowledge interoperations between agents and, at the same time, a fluent flow of different data between different entities.

Various ontologies have been developed to capture particular fields in the manufacturing domain: the OZONE ontology [5] is devoted to constructing scheduling systems, the Enterprise Ontology aims to define the overall activities of an organization [6], the TOVE Ontology focuses on the enterprise modeling [7], the "Machine Shop Information Model" is intended for representing and exchanging machine shop data, initially between manufacturing execution, scheduling, and simulation systems [8], the Process Specification Language (PSL) covers generic process representation common to manufacturing applications [9]. On the other hand, ontologies like MASON [10] or ADACOR [11] could be classified as general-purpose manufacturing ontologies. An interesting standardization initiative has been started by the ONEIDA consortium establishing the framework for both the hardware and the software interoperability at all enterprise levels. Product data, which encapsulates intellectual property along with appropriate semantic information, is collected from the manufacturer and integrators in order to set a searchable repository and ease the work of related intelligent repository agents [12]. Complementary work has been reported by Lopez and Lastra: they merged several ontologies for mechatronic devices reference models (covering both the hardware and the software features) and the IEC 61499 reference model respectively into an ontology for an Automation Objects reference model [13].

However, by now an ontology is missing that will support the usage of design time engineering knowledge for supporting runtime decisions as well as is able to provide system knowledge to agents. The application of agent technology does not bring any advantages if the used agents are not intelligent. Considering ontologies as an intelligent way to manage knowledge, the integration of both technologies brings advantages such as extensibility and communication, and enables agents to agree on the meaning of common concepts they use with any other agent in an open environment [14].

2.2 Representing Design Time Knowledge and Supporting Runtime Decisions

An ontology is a representation vocabulary for a specific domain or subject matter, e.g., production automation. More precisely, it is not the vocabulary as such that qualifies as an ontology, but the (domain-specific) concepts that the terms in the vocabulary are intended to capture [15].

Manufacturing Execution Systems (MES), as a state-of-the-art in production workshops, link plan management and workshop control in an enterprise, which has the advantage to be integrated or interfaced with ERPs. Long [16] constructed a MES ontology which provides a formal specification of the concepts in the MES domain.

The infrastructure of MDA provides an architecture for creating models and meta-models, defining transformations between these models, and managing meta-data. Although the semantics of models are structurally defined by its meta-model, the mechanisms to describe the semantics of a domain are rather limited compared to machine-understandable representations using, e.g., knowledge representation languages like RDF[2] or OWL[3]. In addition, MDA-based languages do not have a know-

[2] Resource Description Framework: http://www.w3.org/RDF/

ledge-based foundation to enable reasoning (e.g., for supporting QA), which ontologies provide [17].

Beyond traditional data models like UML class diagrams or entity relationship diagrams, ontologies provide methods for integrating fragmented data models into a common model without losing the notation and style of the individual models [18]. The idea of using design time engineering knowledge to support runtime decision making have been already introduced by Moser et al. [19]. The authors proposed to collect design models information from production automation systems, such as the workshop layout, customer orders, product tree, or assembly procedures, and integrate them with the run-time information by using an ontology-based approach.

Andreolini et al. proposed to use models and frameworks for supporting runtime decisions in the context of web-based service systems [20]. The problems behind runtime decisions are how to detect significant and non-transient load changes of a system resource and how to predict its future load behavior. The authors described, tested and tuned the two-phase strategy to overcome the problems and integrating the strategy into a framework to support runtime decisions in a cluster web system and in a locally distributed Network Intrusion Detection System.

3 Research Issues and Use Case

In this section, we describe a real-world use case on system adaptation, e.g., to accommodate runtime failures. The use case is based on a Java simulation of an adaptive system that has been validated with a hardware version available at VUT's ACIN lab, the so-called Simulator for Assembly Workshops" (SAW). In the simulation context we collect evidence to which extent a richer and better integrated semantic knowledge base can translate into more accurate faster and cheaper decision making. The general SAW architecture consists of three major layers: the business process layer, the workshop system coordination layer and the machines in the workshop.

SAW simulates complex reconfigurable production automations systems to maximize the overall system output by scheduling sequences of transport and machine tasks over 100 times faster than the actual hardware. In the workshop, each machine has a set of specific functions, e.g., drilling or painting. Production parts are put on pallets and delivered to the machines via conveyor belts. In production automation systems, conveyor belts, junctions, and sensors can be represented by software agents that are working together and form a multi-agent system. By configuring an agent, the behavior of the real hardware can be specified as well. A junction connects two or more conveyor belts and follows the configuration of the software agents; select the correct outgoing conveyor belt for a pallet carrying a work piece. Sensors help the software agents to sense if pallets are in close proximity or help agents counting passing pallets to detect an overloaded conveyor belt and move to a backup strategy.

We defined three scenarios here according to the usage of SAW to solve possible problems that could occur also when using real hardware.

Runtime Decision 1 – Change of Conveyor Directions. The first scenario is related to the way of solving problems caused by conveyors failure. The failures of

[3] Web Ontology Language: http://www.w3.org/2007/OWL

conveyors and especially of conveyors that connect machines to each other, may lead to unreachable machines, hence may result in the failure of the system to produce products at all. One possible option to solve this problem is to change the direction of other unbroken conveyors in order to be used as a substitution of the broken conveyor. By changing other conveyors' direction, it is expected that a new route can be formed; hence the connection between different machines may become available again. The runtime decision that can be taken is to decide which conveyor directions should be changed in order to fulfill the new goal. The operator should also consider which conveyors are available and have a connection to the other machines directly or indirectly in order to identify possible new routes.

Runtime Decision 2 – Machine Reconfiguration. The second scenario in the production automation system that we want to go through is the possibility to reconfigure machines. Usually machines do not only offer one machine function but several different machine functions that can be reconfigured. However it takes time to reconfigure the machine functions inside one machine, e.g., to disassemble and reassemble the machine functions to another machine. If a machine fails, certain machine functions are not available anymore, but there is still exists the possibility to transfer the machine function to another machine which is still working well. In this case, the operator should calculate whether another machine should be reconfigured to offer the needed machine function or not, e.g., to check whether the repair time is smaller than the time needed for reconfiguration, and to check whether there are any products ordered that require the machine function at all.

Runtime Decision 3 – Machine Maintenance Preparation. The third scenario is regarding the maintenance time that is needed by machines in the production automation system. After a certain amount of time running, machines need to undergo maintenance. In order to prepare the maintenance mode, the operator should calculate the number of orders, and therefore the related time needed for all machine functions required for a certain order, which can be produced before the machine needs to go into maintenance mode.

Based on the use case and the three derived exemplary runtime decisions, we derive the following research issues:

RI-1. Efficiency and effectiveness of the proposed engineering ontology. Investigate to what extent the integrated design time and runtime data models do facilitate queries regarding data originating from more than one different data model. Compare the number of queries required for retrieving specific information using the engineering ontology approach with a traditional (e.g., database-based) approach using multiple data sources with potentially heterogeneous data schemata. Can the proposed engineering ontology cope with a potentially very large number of possible solutions for specific scenarios, and how does this affect the answer time of complex queries to the engineering ontology?

RI-2. Scalability of the proposed engineering ontology. Since the evaluated workshop layout is manageably small, investigate the scalability of the proposed engineering ontology approach. Does the ontology area concept [21] support the structuring and therefore the usability of a potentially large and fast-growing engineering ontology? How to support specific stakeholders in working with parts of the engineering ontology, e.g., by providing tool support for specific tasks?

For investigating the research issues we gathered requirements from the use cases in the production automation domain. Based on these use cases we designed the architecture of the engineering ontology and the evaluation regarding the support for the three different runtime decisions.

4 Supporting Runtime Decisions using the engineering ontology

This section describes the architecture and contained design and runtime knowledge of the used engineering ontology, as well as the application of the engineering ontology for supporting the three runtime decisions introduced in the previous section.

4.1. Engineering Ontology Architecture

In this section, we describe the engineering ontology, a set of relevant information elements about components in machine-understandable format using OWL DL ontology syntax. Components can query the engineering ontology at run time to retrieve information for decision making, e.g., enriching and filtering failure information or runtime coordination of machine workloads due to changes of the available machine capacities.

The engineering ontology provides a place for storing design-time information that seems valuable for supporting run-time decisions of components, especially in the case of handling failures or unplanned situations (but not transformed into run-time code or configuration to limit their complexity).

Components can query the engineering ontology at run time with query languages like SPARQL[4] (SPARQL Protocol and RDF Query Language), which provide the components with the full expressive power of ontologies, including the ability to derive new facts by reasoning. In addition, components can feed back interesting observations into the run-time information collection of the engineering ontology and therefore help to improve the design-time models (e.g., by improving estimated process properties with analysis of actual run-time data) and/or check the information based on a certain set of assertions. Furthermore, valuable deployment information can also be stored in the engineering ontology in order to support and enhance for further deployments.

Figure 2 shows the three different layers involved in SAW: a) the *business layer* for production planning to fulfil customer orders by assigning optimal work orders to the workshop; b) the *workshop layer* for coordinating the complex system of transport elements and machines to assemble smaller basic products into larger more comprehensive products according to the work orders; and c) the *operation layer* for monitoring the individual transport system elements and machines to ensure their contributions to the workshop tasks. Those three layers are divided into two parts based on the time those layers worked on, namely design time (development) and run time (usage).

[4] www.w3.org/TR/rdf-sparql-query/

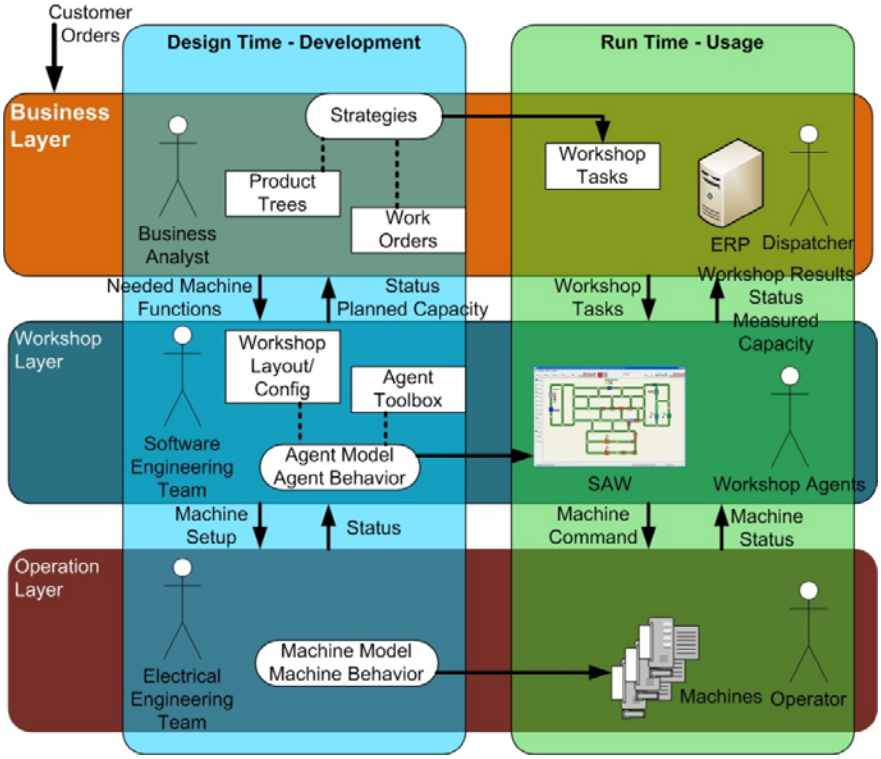

Figure 2. Engineering Ontology Overview.

In prior work [21], we proposed a data modelling approach that helps structuring large ontologies using ontology building blocks, so-called *"Ontology Areas"*. An ontology area is a part of an ontology, which is meaningful for a stakeholder and which helps ontology users to manage a complex ontology. The combination of all needed ontology area represents the overall ontology for supporting the original engineering process. An ontology area is a subset of an ontology as a building block that can solve a certain task. The ontology can be broken into ontology areas based on several aspects, for example by time, volatility, layer and roles. Figure 2 shows the breakdown of ontology into several ontology areas based on the stakeholder layers (business, workshop, operation) and the time the models are mostly used (design time and run time). Some parts of the data mode are much more volatile than others, e.g., run-time process measurements compared to design-time workshop layout.

4.2 Runtime Decision 1 – Change of Conveyor Directions

A Change-Direction-Algorithm (CDA) is used to handle a breakdown of conveyors which might lead to unreachable destinations (machines). The CDA is able to find the best stable configuration of the transport system by changing the directions of specific conveyors [22].

Each component of the transport system (e.g. conveyor belts, nodes, etc.) is represented and controlled by a corresponding Automation Agent [23] — an autonomous semantic entity responsible for the maintenance of the local data described in its world model. Besides the Automation Agents we also introduced the Contact Agent (CA) [24], which is created at the start-up of the system and is always active. Its main responsibilities are to supervise the functionality of the system and in the case that one part of the system collapses this agent considers its influence on the system performance and, if significant, undertakes particular steps in order to bring the system back into the optimal state.

The system uses the CDA and reacts on failures as follows:

1) The hardware of the system detects the failure using sensors. The low-level control informs the corresponding high-level control through the low-level communication interface [25].
2) Based on the given information the agent updates its knowledge base and informs all related agents about the detected failure. Each node has to re-calculate its routing table.
3) Furthermore, the agent also informs the CA, which is responsible for the overall system functionality.
4) The CA starts the CDA and compares its results with the actual system state.
5) In the case that the CDA recommends a new configuration, the CA updates its ontology, requests conveyor agents of concerned conveyors to change directions, and informs node agents to update their system representation.
6) Each node agent will recalculate and update its routing table. Having accurate information and an up-to-date world model of the system is of primary importance for nodes. Due to their role to receive pallets coming from input conveyors and —according to their destinations — to route them to the appropriate output conveyors.

Listing 1 shows the Prolog code (we use the Prolog notation for simplicity reasons) for detecting routes between two machines. There are 4 conveyors that connect machine A and machine B, machine B and junction C, junction C and junction D, and machine A and junction D. The connection between two nodes is defined bidirectional, i.e., whether there is a conveyor from a node to another node or vice versa. The routes from a node to another node are specified either as a connection directly from the second node to the first node (backtrack) or if there is a connection from the first node to the next node on the list of the nodes on the route to second node. This is done recursively. Hence the route from machine A to machine B can be discovered by using the query *route(machineA,machineB,Y).* This query returns two results, namely machine A → junction D → junction C → machine B and machine A → machine B, which means if the conveyor from machine A to machine B is failing, there is the possibility to use an alternative route.

This query requires design time workshop layout information regarding conveyors that are connected to machines or to other conveyors to check possible routes and change conveyor directions if required.

```
conveyor('machineA','machineB').
conveyor('machineB','junctionC').
conveyor('junctionC','junctionD').
conveyor('junctionD','machineA').

connect(NodeX,NodeY) :- conveyor(NodeX,NodeY),
  conveyor(NodeY,NodeX).

route(Node1,Node2,Way) :- go(Node2,Node1,[],Way).

go(Node,Node,Oldway,[Node|Oldway]).
go(Node1,Node2,Oldway,Way) :- connect(Node1,ANode),
  member(ANode,Oldway),
  go(ANode,Node2,[Node1|Oldway],Way).

?- route(machineA,machineB,Y).
Y = [machineA, junctionD, junctionC, machineB].
Y = [machineA, machineB].
```

Listing 1. CDA Query for detecting routes between two machines.

4.3 Runtime Decision 2 – Machine Reconfiguration

The second scenario deals with the possible reconfiguration of machines. Since there may be available modular assembly machines that can perform more than one different machine function, there always is the option to reconfigure a certain machine; either to provide a machine function in case of a failure of the original machine providing that function, or to provide an additional instance of this machine function to increase the throughput. However, this reconfiguration does not only may take essential time, but may also be not required at all, since it depends on the type and number of (future) orders to be produced. All of this information needs to be taken into consideration before machine reconfiguration should take place, and since these considerations may become quite complex, automation support is required for an efficient and effective machine reconfiguration process.

Listing 2 shows the Prolog code for calculating the reconfiguration time of a machine and comparison with the repair time needed to repair the failed machine providing the required machine function, so the operator can decide whether to reconfigure an alternative machine to provide the required machine function or wait for the repair of the failed machine. We have the design time information regarding the time needed to disassemble the machine function mf1 from machine A and also regarding the time required to assemble machine function mf1 to machine B. Hence, the reconfiguration time is calculated as sum of disassembly and assembly time of machine function mf 1. By applying the query *reconfigure(mf_1)*, we can check whether the time required to reconfigure an alternative machine is smaller than the time needed to repair the failed machine, which is true for the used exemplary values.

This query requires design time machine configuration information regarding the time required to disassemble and assemble machine function from certain machines, as well information regarding standard repair times of machines. Then we can com-

pare the time needed to reconfigure an alternative machine to provide a required machine function or to repair failed machine.

```
disassembly('machineA','mf_1',100).
assembly('machineB','mf_1',200).

reconfiguration_time(MachineFunction,Time) :-
  disassembly(_,MachineFunction,DisassemblyTime),
  assembly(_,MachineFunction,AssemblyTime),
  Time is DisassemblyTime + AssemblyTime.

repair_time('mf_1',500).

reconfigure(MachineFunction) :-
  reconfiguration_time(MachineFunction,TimeA),
  repair_time(MachineFunction,TimeB),
  TimeA =< TimeB.

?- reconfigure(mf_1).
true.
```

Listing 2. Query for calculating the reconfiguration time of a machine.

4.4 Runtime Decision 3 – Machine Maintenance Preparation

A third relevant run-time decision in the production automation scenario is the decision when to perform machine maintenance tasks in order to keep a certain minimum level of production output. This decision could also be taken during design time, resulting in a decreased ability to react to new or changing environment conditions (e.g., failures, reconfiguration). Since system flexibility supports operational efficiency in production automation, the decision when to perform machine maintenance tasks (e.g., cleaning, refurbishment) and the preparations for these tasks (i.e., emptying the machine buffers) could be taken by the machines themselves taking into account the state of other machines and workshop environment conditions. The idea is to coordinate the maintenance tasks of a set of related machines to minimize the impact on the overall production process. This planned maintenance should also be reported to a controlling system (e.g., an ERP system) in order to allow in-time reaction to the future capacity changes.

Listing 3 shows the Prolog code for calculating the total runtime of a machine. By using this code, the operator can calculate the total time to run all machine functions of a specific machine. The operator can check whether the total time is not exceeding the maximal runtime of the machine. By using the query *overuse('machineA')*, we can check whether the machine A overuses the maximal runtime allowed by the simulation workshop. In the used example, the total time is still below the maximal allowed runtime.

This query requires design time machine configuration information regarding the different times needed to run each machine function of particular machines.

```
run('machineA','mf_1',50).
run('machineA','mf_2',75).
run('machineA','mf_3',100).

list_of_run_time(List,X) :-
  findall(T,run(X,MF,T),List).

list_sum([], 0).

list_sum([Head | Tail], TotalSum) :-
  list_sum(Tail, Sum1),
  TotalSum is Head + Sum1.

total(X,Time) :-
  list_of_run_time(List,X),
  list_sum(List,Time).

maximal_runtime('machineA',300).

overuse(MachineName) :-
  total(MachineName,TimeA),
  maximal_runtime(MachineName,TimeB),
  TimeA > TimeB.

?- overuse('machineA').
false.
```

Listing 3. Query for calculation the total runtime of a machine.

5 Discussion and Conclusion

Engineers of complex software-intensive systems such as the “Simulator of Assembly Workshops” (SAW) system, who want to adapt the system at runtime, need information from software models that reflect dependencies between components at design and run time, e.g., the workshop layout, customer orders and assembly procedures that translate into needs for machine function capacities over time; and the coordination of tasks for redundant machines in case of a failure. Without an integrated view on relevant parts of both design-time and run-time models inconsistencies from changes and their impact are harder to evaluate and resolve between design and run time. Better integrated engineering knowledge can improve the quality of decisions for run-time changes to the system, e.g., better handling severe failures with predictable recovery procedures, lower level of avoidable downtime, and better visibility of risks before damage occurs.

In this paper, we presented an approach to improve support for runtime decision making using an engineering ontology. This engineering ontology provides a better integrated view on relevant engineering knowledge in typical design-time and runtime models, which were originally not designed for machine-understandable integration. We illustrated and showed the feasibility of the engineering ontology approach with three exemplary scenarios (change of conveyor directions, machine reconfiguration

and machine maintenance preparation) from the production automation domain, an extensive empirical evaluation will be performed in our future research work.

Based on this evaluation, we addressed the following research issues:

RI-1. Efficiency and effectiveness of the proposed engineering ontology. The engineering ontology provides a better integrated view on relevant engineering knowledge contained in typical design-time and run-time models in machine-understandable form to support runtime decisions. Another benefit is the possibility to define assertions in the engineering ontology which are checked based on the run-time information input of the running components. Further, the quality of information presented to an operator is improved since all information both from design-time as well as from run-time is available, leading to more intelligent run-time analysis and decision support.

RI-2. Scalability of the proposed engineering ontology. Typically, the engineering ontology can become very large and complex compared to the basic data model (such as used in a data base to automate run-time processes) if they include several aspects on a domain and some parts of the data model are volatile. In this paper, we proposed to use a data modeling approach based on ontology building blocks, so-called "Ontology Areas" [21], which allow solving tasks with smaller parts of the overall ontology. Ontology Areas also improve the efficiency of data collection task for decision making by lowering the cognitive complexity for designers and users of the ontology.

Future Work. While the engineering ontology can be seen as a comprehensive ontology, which stores and uses engineering knowledge both at design time and run time, more manageable, their application needs the effort of designers for structuring the overall ontology and for building task-specific smaller ontologies. Thus we will conduct empirical studies on the effort needed to design and use the engineering ontology. Future work could include human-subject experiments to assess complexity and efficiency more rigorously.

Acknowledgments. This work has been supported by the Christian Doppler Forschungsgesellschaft and the BMWFJ, Austria; and also by the FIT–IT: Semantic Systems program, an initiative of the Austrian federal ministry of transport, innovation, and technology (bm:vit) under contract FFG 815132.

References

1. Merdan, M., Moser, T., Wahyudin, D., Biffl, S.: Performance evaluation of workflow scheduling strategies considering transportation times and conveyor failures. International Conference on Industrial Engineering and Engineering Management. IEEE (2008) 389-394
2. Jennings, N.R., Bussmann, S.: Agent-based control systems: Why are they suited to engineering complex systems? Control Systems Magazine, IEEE **23** (2003) 61-73
3. Sycara, K.P.: Multiagent systems. AI magazine **19** (1998) 79-92
4. Kulvatunyou, B., Cho, H., Son, Y.J.: A semantic web service framework to support intelligent distributed manufacturing. Int. J. Know.-Based Intell. Eng. Syst. **9** (2005) 107-127
5. Smith, S.F., Becker, M.A.: An ontology for constructing scheduling systems. Working Notes of 1997 AAAI Symposium on Ontological Engineering. AAAI Press (1997)

6. Uschold, M., King, M., Moralee, S., Zorgios, Y.: The Enterprise Ontology. Knowl. Eng. Rev. **13** (1998) 31-89
7. Fox, M.S., Barbuceanu, M., Gruninger, M.: An organisation ontology for enterprise modeling: Preliminary concepts for linking structure and behaviour. Computers in Industry **29** (1996) 123-134
8. McLean, C.R., Lee, Y.T., Shao, G., Riddick, F.: Shop data model and interface specification. NISTIR 7198. National Institute of Standards and Technology (2005)
9. Grüninger, M., Kopena, J.B.: Planning and the Process Specification Language. WS2 ICAPS 2005 (2005) 22-29
10. Lemaignan, S., Siadat, A., Dantan, J.-Y., Semenenko, A.: MASON: A Proposal For An Ontology Of Manufacturing Domain. IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications. IEEE Computer Society (2006) 195-200
11. Leitão, P., Restivo, F.: ADACOR: A holonic architecture for agile and adaptive manufacturing control. Computers in Industry **57** (2006) 121-130
12. Vyatkin, V.V., Christensen, J.H., Lastra, J.L.M.: OOONEIDA: an open, object-oriented knowledge economy for intelligent industrial automation. Industrial Informatics, IEEE Transactions on **1** (2005) 4-17
13. Lopez Orozco, O.J., Martinez Lastra, J.L.: Using semantic web technologies to describe automation objects. International Journal of Manufacturing Research **1** (2006) 482-503
14. González, E.J., Hamilton, A.F., Moreno, L., Marichal, R.L., Muñoz, V.: Software experience when using ontologies in a multi-agent system for automated planning and scheduling. Softw. Pract. Exper. **36** (2006) 667-688
15. Chandrasekaran, B., Josephson, J.R., Benjamins, V.R.: What are ontologies, and why do we need them? IEEE Intelligent Systems and Their Applications **14** (1999) 20-26
16. Long, W.: Construct MES Ontology with OWL. ISECS International Colloquium on Computing, Communication, Control, and Management (CCCM '08), Vol. 1 (2008) 614-617
17. Baclawski, K., Kokar, M.K., Kogut, P.A., Hart, L., Smith, J., Letkowski, J., Emery, P.: Extending the Unified Modeling Language for Ontology Development. International Journal of Software and Systems Modeling (SoSyM) **1** (2002) 142-156
18. Hepp, M., De Leenheer, P., De Moor, A., Sure, Y.: Ontology Management: Semantic Web, Semantic Web Services, and Business Applications. Springer-Verlag (2007)
19. Moser, T., Schatten, A., Sunindyo, W.D., Biffl, S.: A Run-Time Engineering Knowledge Base for Reconfigurable Systems. Technical Report (available at: http://bit.ly/hOr9Tf), Vienna, Austria (2009)
20. Andreolini, M., Casolari, S., Colajanni, M.: Models and framework for supporting runtime decisions in Web-based systems. ACM Trans. Web **2** (2008) 1-43
21. Biffl, S., Sunindyo, W.D., Moser, T.: Bridging Semantic Gaps Between Stakeholders in the Production Automation Domain with Ontology Areas. 21st International Conference on Software Engineering & Knowledge Engineering (SEKE 2009). KSI (2009) 233-239
22. Koppensteiner, G., Merdan, M., Hegny, I., Weidenhausen, G.: A change-direction-algorithm for distributed multi-agent transport systems. Mechatronics and Automation, 2008. ICMA 2008. IEEE International Conference on (2008) 1030-1034
23. Vallée, M., Kaindl, H., Merdan, M., Lepuschitz, W., Arnautovic, E., Vrba, P.: An automation agent architecture with a reflective world model in manufacturing systems. IEEE International Conference on Systems, Man and Cybernetics. IEEE Press (2009) 305-310
24. Merdan, M.: Knowledge-based Multi-Agent architecture applied in the assembly domain (Available at:http://www.ub.tuwien.ac.at/diss/AC05040230.pdf). PhD Thesis, VUT (2009)
25. Merdan, M., Lepuschitz, W., Hegny, I., Koppensteiner, G.: Application of a communication interface between agents and the low level control. Autonomous Robots and Agents, 2009. ICARA 2009. 4th International Conference on (2009) 628-633