Knowledge-based Runtime Failure Detection for Industrial Automation Systems

Martin Melik-Merkumians, Thomas Moser, Alexander Schatten, Alois Zoitl and Stefan Biffl

Christian Doppler Laboratory for Software Engineering Integration for Flexible Automation Systems
Vienna University of Technology, Austria
{firstname.lastname}@tuwien.ac.at

Abstract. Engineers of complex industrial automation systems need engineering knowledge from both design-time and runtime engineering models to make the system more robust against normally hard to identify runtime failures. Design models usually do not exist in a machine-understandable format suitable for automated failure detection at runtime. Thus domain and software experts are needed to integrate the fragmented views from these models. In this paper we propose an ontology-based engineering knowledge base to provide relevant design-time and runtime engineering knowledge in machine-understandable form to be able to better identify and respond to failures. We illustrate and evaluate the approach with models and data from a real-world case study in the area of industrial automation systems. Major result was that the integrated design-time and runtime engineering knowledge enables the effective detection of runtime failures that are only detectable by combining runtime and design-time information.

1 Introduction

Complex industrial automation systems 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 runtime with a range of configuration options. The educational process plant is used to simulate complex industrial batch processes (like refineries, breweries, or pharmaceutical plants). It consists of two tanks holding the process fluids. The liquid level of the tanks are checked by several analog and digital sensors. The lower tanks also contains a pump which either transports the process fluid into the upper tank, or is used to mix up the process fluid in the lower tank. Also the lower tank contains a heater and a temperature sensor to heat the process fluid to specified temperatures. Figure 1 shows on the left hand side an image of the real system, while the right hand side of the figure displays a simplified version of the underlying data model of the educational process plant. Additionally, some

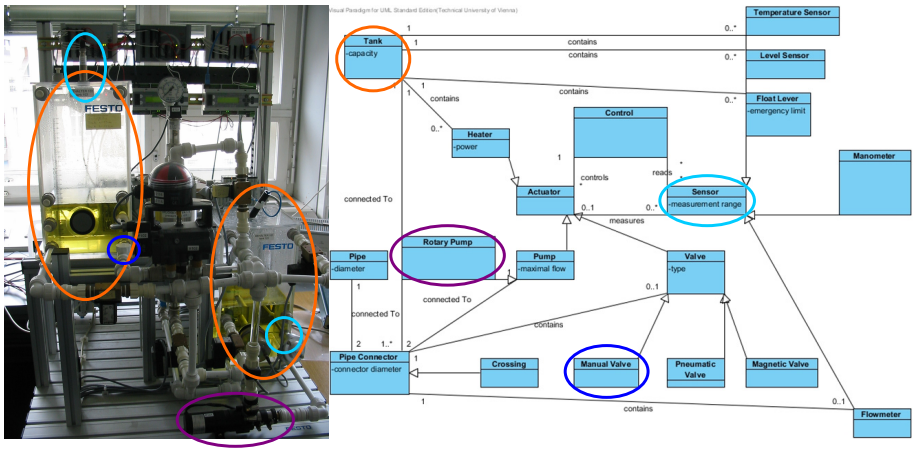

Fig. 1. Educational Process Plant - real system and underlying data model

of the data models elements have been colored in the same color as their real-world representation in the image of the educational process plant in order to show the links between real-world system and underlying data model.

Engineers, who want to detect failures at runtime which can not be compassed by analyzing singular sensor values or failures at sensor-less components (e.g. broken actuators which are not monitored by a sensor), need information from software models that reflect dependencies between components at design and runtime, e.g., the workshop layout, recipes and production procedures. During development design-time software models (representing electrical, mechanical, and software engineering models), like data-oriented models (e.g., EER models or P&ID[1] (Piping and Instrumentation) diagrams [8]) or work flow-oriented models (e.g., sequence diagrams or state charts) are the basis to derive runtime models but are often not provided in machine-understandable format to reflect on failures at runtime, i.e., the knowledge is kept in 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 complex industrial automation systems (which like the educational process plant consist of components defined by general design-time behavior, derived runtime configuration, and runtime specific behavior enactment) more robust against normally hard to identify runtime failures. QA people could

[1] Industrial standard for P&IDs: IEC 61346: Industrial systems, Installations and Equipment and Industrial Products Structuring Principles and Reference Designations

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 runtime come 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 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 runtime. Better integrated engineering knowledge can improve the quality of decisions for runtime 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. As shown in [11], with the help of ontologies and reasoning most of these problems can be addressed.

In this paper we present an approach to improve support for runtime decision making with an ontology: a domain-specific engineering knowledge base (EKB) that 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 EKB 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 an 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 ontology-based approach with two types of runtime failure (RTFs) from a real-world use case study in the area of industrial automation systems. Major result was that the integrated design-time and runtime engineering knowledge enables the effective detection of normally hard to identify runtime failures.

In the remainder of the paper we survey relevant engineering models for their contributions and limitations to support runtime decision making; we describe a real-world case on runtime failure detection for collecting evidence to which extent richer and better integrated semantic knowledge can translate into better decision making.

2 Evolution of Engineering Models towards Runtime System Analysis and Adaptation

Engineering models have evolved from means to structure complex domains and designs at design time towards model-driven approaches that bring domain information closer to implementation and runtime. However, systems that are designed for adaptation at runtime need more advanced approaches to provide relevant and accurate engineering knowledge to guide runtime system analysis and adaptation.

Structuring design complexity. Models are used on various levels in software engineering: Data models like entity relationship (ER) diagrams originate in the late 1970s [5]. With the Unified Modeling Language (UML) [3] a standardized set of diagrams and modeling techniques were introduced for object-oriented design. However, these models are mostly used initially during the design time of a project and get seldom adapted to changes during implementation or operation.

Connecting design and implementation. There are tool providers, who claim to support round trip engineering from design models to source code and back. However, a stronger and more consistent integration between the design models and the implementation phase artifacts comes from the Model-Driven Architecture (MDA), Model-Driven Development (MDD) [16], and Model-Driven Configuration management. There are several interesting aspects about MDD: The models develop from high-level abstractions to concrete code over several intermediate steps. The initial model can be a general UML model or a domain-specific language model. In MDD (opposed to earlier modeling approaches) the model is used also in the implementation phase, i.e., used to create platform-specific code, but is not used at runtime. There also exist approaches for using MDA for the engineering of automation systems. Melik-Merkumians et al. [10] present an approach that separates between logical control applications and the plant model. The logical control application models the intended behavior of the control application in a target independent way. The plant model defines the control devices, their abilities, and their interconnections (e.g., communication system). By mapping both models together the control code executed in the control devices with their hardware specific parameters can be generated automatically.

Connections between design, implementation, and runtime. Traditional software engineering approaches mostly focus on the development phase and see configuration management (CM) as a support task. However, CM is an example model that is valuable at design time, implementation, deployment, and runtime. Some approaches thus suggest including CM and application life cycle management (ALM) into the MDD concept [6].

Runtime needs of distributed reconfigurable software-intensive systems. Ahluwalia et al. [1] observe a shift from "monolithic to highly networked, heterogeneous, interactive systems" that has led to a "dramatic increase in both development and system complexity", where at the same time the "demands for safety, reliability, and other qualitative attributes have increased across application domains." Oreizy et al. [14] additionally mention the necessity of "runtime evolution" of modern multi-user, distributed systems. Today many deployed applications gradually evolve over time (ideally without downtime for users) rather than undergo "big bang" version updates. Examples for such applications are e-Commerce Services like Online-Banking applications as well as most "Web 2.0" applications. The problem gets particularly critical in domains like distributed real-time and embedded systems as in automotive and production automation industry applications.

Feedback of runtime experience to design. An underlying trend is to bring development activities closer to the runtime environments, i.e., using data from the deployed system for engineering purposes (e.g., QoS parameters). Additionally, we observe more intensive research activities [17] to design and apply MDD where the models do not stop at development but also support the runtime environment of the system. Recent research investigates mechanisms towards automatic runtime failure detection [17] and ultimately self-healing systems. Garlan et al. [6] describe model-based approaches for self-healing autonomous systems with a similar idea: remove the traditional separation between system creation/-modification and runtime environment with an integrated approach. The authors particularly point out that system-internal exception handling and configuration (hence not always easy to change) is problematic in many modern systems as they are designed to run continuously, and also updates and reconfiguration should be done without system shutdown.

Ontologies for connecting design-time and runtime data models. 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 [4]. The infrastructure of MDA provides architecture for creating models and meta-models, defining transformations between these models, and managing meta-data. Although the semantics of a model is structurally defined by its meta-model, the mechanisms to describe the semantics of the 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 knowledge-based foundation to enable reasoning (e.g., for supporting QA), which ontologies provide [2]. 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 [7]. The usage of ontologies for knowledge representation and sharing, as well as for and high-level reasoning could be seen as a major step towards the area of agent-based control solutions [13]. Exploitation of semantics and ontologies in the area of agent-based industrial systems has become one of the major research areas in the last few years, primarily because of the success and promotion of semantic web technologies to enable better communication between machines and people [15]. Ontologies are considered here as an essential technology for semantic web development guaranteeing data and information interoperability in heterogeneous and content-rich environments [12].

3 An Integrating Engineering Knowledge Base

In this section, we introduce the Engineering Knowledge Base (EKB), a set of relevant information elements about components in machine-understandable

[2] Resource Description Framework: http://www.w3.org/RDF
[3] Web Ontology Language: http://www.w3.org/2007/OWL

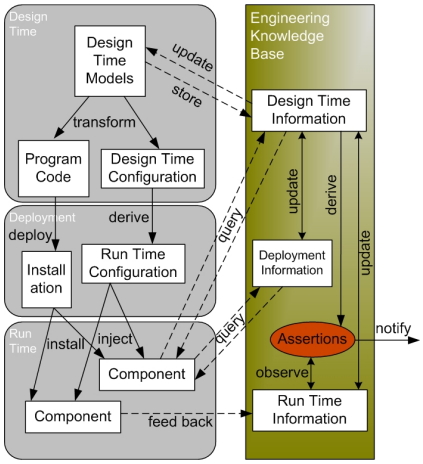

Fig. 2. An Engineering Knowledge Base in Context

format using ontology syntax. Components can query the EKB at runtime to retrieve information for detecting normally hard to identify failures or implausibilities of the current system.

Figure 2 illustrates 3 major phases in the life cycle of complex industrial automation systems:

1. Design time: Models that describe the automation system layouts, the recipes of the manufactured products, etc. are transformed into executable program code and design-time configuration instructions.
2. In the Deployment phase, the executable program code is deployed into installable packages and the runtime configuration is derived from the design-time configuration.
3. At runtime the deployed program code for system operation gets installed to a set of components and the runtime configuration gets injected into these components.

This architecture has proven effective for systems whose properties change seldom, since the effort needed for transformation, deployment, and injection is considerable.

However, typical complex industrial automation systems also suffer from failures, e.g., if some components fail or become unavailable. To support failure detection, the components need to be able to perform decisions at runtime, since a complete new iteration of model transformation, program code deployment, and configuration injection would take too long. A major challenge of runtime failure detection is to provide access to relevant design-time information that is usually stripped away during transformation for efficiency reasons. The Engineering Knowledge Base (EKB) provides a place for storing design-time information that seems valuable for supporting runtime failure detection of components, especially

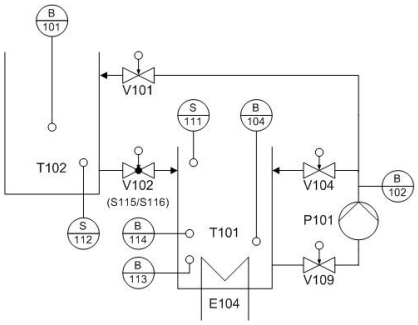

Fig. 3. P&ID of the Educational Process Plant

in the case of handling failures or unplanned situations (but not transformed into runtime code or configuration to limit their complexity).

Components can query the EKB at runtime with semantic web query languages like SPARQL[4] (SPARQL Protocol and RDF Query Language) or SWRL[5] (Semantic Web Rule Language), which provide to the components 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 runtime information collection of the EKB and therefore help to improve the design-time models (e.g., by improving estimated process properties with analysis of actual runtime data) and/or check the information based on a certain set of assertions. Furthermore, valuable deployment information can also be stored in the EKB in order to support and enhance for further deployments.

Based on the design-time information, it is possible to define a set of runtime assertions in the EKB. These runtime assertions observe the runtime information fed back into the EKB and can notify a specific role or system if the violation of an assertion has been detected.

4 Real-World Use Case and Results

In this section we describe two real-world failure use cases which could lead to equipment defects and/or waste of used production material. Especially failures which are hard to identify by traditional means can lead to subsequent equipment damage and decrease of process performance. Therefore the identification of such failures is of great importance. We will show that such failures can be identified by usage of the EKB. We will also see that the EKB can provide soft sensors by simple (and therefore easy to implement in PLC (programmable logic controller) programs) reasoning of process data. The runtime assertions will be given in pseudo code samples. The assertions can be evaluated periodically, as it would be in a traditional PLC environment, or in regard to certain events, like the

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

change of a sensor value. As long as all needed sensor inputs and PLC outputs are available and stable, the current system status is correctly mapped.

The model we use to demonstrate our failure use cases is an educational process plant in the Odo-Struger-Laboratory[6]. Figure 3 shows the P&ID (Piping and Instrumentation Diagram) of the tank model. It consists of two tanks, T101 and T102, which are on different height levels. The upper tank (T102) contains an ultrasonic level sensor (B101) which measures the distance of the liquid surface to the upper tank closure, and a float lever which represents a critical low level of the liquid. The lower tank (T101) consists of three float levers (a critical upper level (B114), a lower level (B104), and a critical lower level lever (B113)), a heater (E104), and a temperature sensor (S111). The two tanks are connected by several pipes which are opened or closed by several valves (the symbols marked with V). Valves are usually not equipped with sensors to determine their current state, due to financial reasons. The liquid transportation from the lower to the upper tank is done by the pump P101, The actual flow caused by the pump is measured by the flow meter (B102). Even as this process plant is an educational model it represents a typical plant configuration in the process industry. Listing 1 shows some of the EKB's triples defining the tank T101 with blabla. For a complete listing of the EKB's triples representing the P&ID model elements please refer to [9].

Listing 1. Example of EKB triples representing P&ID model elements

```
<Tank rdf:ID="T102">
  <contains>
    <Float_Lever rdf:resource="#LS-102"/>
  </contains>
  <contains>
    <Level_Sensor rdf:resource="#LIC-102"/>
  </contains>
  <connected_To>
    <Pipe_Connector rdf:resource="#PC-B102-1"/>
  </connected_To>
  <connected_To>
    <Pipe_Connector rdf:resource="#PC-B102-2"/>
  </connected_To>
  <capacity rdf:datatype="xml:#float">1000.0</capacity>
</Tank>
```

RTF-1: Undetected valve V101 failure in the pump pipe section. In this scenario the liquid in tank T101 shall be pumped into tank T102. Therefore valve V104 must be closed and the valves V109 and V101 must be opened. We now assume that one or both valves that should be open are defect and therefore unable to get into the open position. As the control has no means to control the position of the valves the pump P101 starts. Such a situation can lead to the destruction of the pump, as it is either pumping against the impregnable

[6] The industrial automation systems laboratory of the Automation and Control Institute, Vienna University of Technology

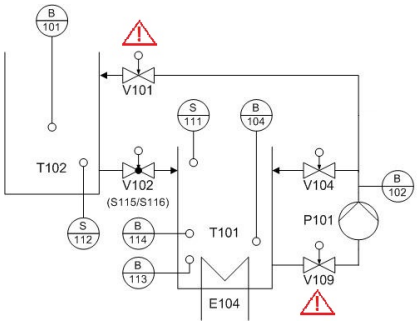

Fig. 4. P&ID showing RTF-1: Undetected valve V101 failure

resistance of valve V101, or it will be soon out of water as valve V109 inhibits further water supply, which leads to destruction by for example rotary pumps.

To avoid such equipment failure it is common to use a flow meter (B102 in Figure 4), which can be used compare the actual flow with the anticipated flow value. If those two values differ too much the pump can be put into an emergency shut down. But if the flow meter is defect or the flow meter is omitted for cost reduction, the failure can not be detected anymore. Such failures can lead to huge costs due to equipment loss, material waste, and additional down times and should therefore be avoided. Engineering knowledge can be used to provide a simple rule to check the functionality of the pump system. As the upper tank T102 has a level sensor it can be checked if the liquid level of the upper tank is rising if the pump is activated. A more correct model would additionally be able to determine if the level rising is proportional to the actual pump power. Through consequent usage of engineering knowledge such "soft sensors" (instead of a real sensor the value has been calculated from the process model) can be created. By comparison of the measured and calculated flow value the system can also determine if the flow meter or the pump is behaving as expected. Listing 2 shows this query in pseudo code. For a complete listing of the query in full SPARQL syntax please refer to [9].

Listing 2. Pseudo code query for detecting failures of V101

```
while (P101.isActive())
  for (B102.getSensorEvent() as x)
    for (B102.getSensorEvent() as y)
     if (x.getTimestamp() < y.getTimestamp())
       if ( NOT (y.getLevel() > x.getLevel()) )
         << raise alarm >>
```

RTF-2: Leakage in valve V102 with subsequent material loss. Such a failure can lead to enormous costs due to material waste, lower quality end product (perhaps even to low to sell), and wasted production time. Especially the wasted production time is costly as process engineering processes (like refining and chemical processes in general) usually need long times to be completed. Such

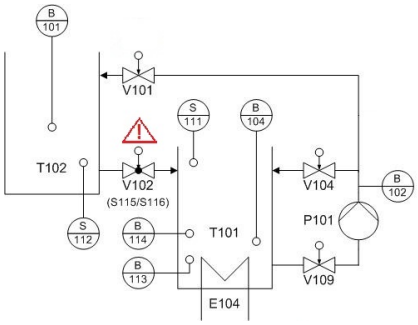

Fig. 5. P&ID showing RTF-2: Leakage in valve V102

failures are hard to detect by conventional alarming mechanisms, as traditional alarms are only issued if a process value reaches a critical threshold.

In this scenario we assume that valve V102 is leaky (Figure 5), which is why the process material in tank T102 is slowly petering out. In an ordinary control program would only wait for a longer time until the liquid threshold defined in the program is attained. There is no way to detect the leakage by simple sensor or alarm processing, only well grounded personnel in the control room can detect such a failure in a traditional PLC system.

Listing 3. Pseudo code query for detecting leakage in valve V102

```
while (NOT (P101.isActive()) )
  if (V102.isClosed())
    for (B102.getSensorEvent() as x)
      for (B102.getSensorEvent() as y)
        if (x.getTimestamp() < y.getTimestamp())
          if ( NOT (y.getLevel() == x.getLevel() )
            << raise alarm >>

while (P101.isActive())
  if (V102.isClosed() AND V104.isClosed())
    if (V109.isOpen() AND V101.isOpen())
      for (B102.getSensorEvent() as x)
        for (B102.getSensorEvent() as y)
          if (x.getTimestamp() < y.getTimestamp())
            if ( NOT (y.getLevel() =
              x.getLevel() * P101.getPumpPower()) )
                << raise alarm >>
```

Once again the usage of engineering knowledge leads to rules capable to detect this failure. The first rule is, if the pump P101 is not pumping and valve V102 is closed, then the fluid level of tank T102 has to be constant. If there is a leakage in the valve then the level would sink. Assuming that the flow meter B102 is working correct the second rule is, if the valves V102 and V104 are closed,

and the valves V109 and V101 are open, and the pump P101 is pumping, then the fluid level of tank T102 must rise proportional to the actual pump power.

The first rule is capable of detecting a leakage in tank T102, but the second rule can only state that some condition is violated. The solution to this problem is rather easy and efficient. If the second rule fires than stop pump P101 and reevaluate rule one for leakage detection. Another benefit of this behavior is that the material loss is limited to the content of tank T102 or stopped altogether. Listing 3 shows this query in pseudo code. For a complete listing of the query in full SPARQL syntax please refer to [9].

5 Summary and Further Work

In this paper we described an ontology-based approach to provide relevant design-time and runtime engineering knowledge stored in a so called Engineering Knowledge Base (EKB). The EKB provides a better integrated view on relevant engineering knowledge contained in typical design-time and runtime models in machine-understandable form to support runtime failure detection. This approach is useful in the industrial automation domain, and can more generally be used for other (distributed) engineering systems. We illustrated our approach with two use cases of runtime failure detection from a real-world case study in the area complex industrial automation systems. The use cases identified the needs for a complex decision system for failures that are only detectable by combining sensor information with engineering knowledge and showed a feasible query based approach suitable for the task.

Major result of the evaluation of the proposed EKB approach was the possibility to define assertions in the EKB which are checked based on the runtime information input of the running components. This can be seen as external Quality Assurance (QA) without interfering with the original production system and therefore it has proven to be easier to enrich existing applications without the need to make changes to legacy systems (smoother migration path). Further, the quality of information presented to an operator is improved since all information both from design-time as well as from runtime is available, leading to more intelligent runtime analysis and decision support.

Further important practical issues are to investigate the effort needed to import the data from the relevant models into the Engineering Knowledge Base and improving the performance of data access and reasoning at runtime. The use of assertions for checking QoS parameters like system throughput is open to further research too.

Acknowledgment

This work has been supported by the Christian Doppler Forschungsgesellschaft and the BMWFJ, Austria. This work has been partially funded by the Vienna University of Technology, in the Complex Systems Design and Engineering Lab.

References

1. Ahluwalia, J., Krger, I.H., Phillips, W., Meisinger, M.: Model-based run-time monitoring of end-to-end dead-lines. In: 5th ACM international conference on Embedded software (EMSOFT '05). pp. 100–109. ACM (2005)
2. 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(2), 142–156 (2002)
3. Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modeling Language Reference Manual. Addison-Wesley (1999)
4. Chandrasekaran, B., Josephson, J.R., Benjamins, V.R.: What are ontologies, and why do we need them? IEEE Intelligent Systems and Their Applications 14(1), 20–26 (1999)
5. Chen, P.P.S.: The entity-relationship model - toward a unified view of data. ACM Transactions on Database Systems (TODS) 1(1), 9–36 (1976)
6. Garlan, F., Lopez de Vergara, J., Fernandez, D., Munoz, R.: A model-driven configuration management methodology for testbed infrastructures. In: IEEE Network Operations and Management Symposium NOMS 2008. pp. 747–750. IEEE (2008)
7. Hepp, M., De Leenheer, P., De Moor, A., Sure, Y.: Ontology Management: Semantic Web, Semantic Web Services, and Business Applications. Springer-Verlag (2007)
8. Love, J.: Process Automation Handbook: A Guide to Theory and Practice. Springer-Verlag London (2007)
9. Melik-Merkumians, M., Moser, T., Schatten, A., Zoitl, A., Biffl, S.: Knowledge-based runtime failure detection for industrial automation systems. Tech. rep., Christian Doppler Laboratory for Software Engineering Integration for Flexible Automation Systems Vienna University of Technology, Austria (2010), http://cdl.ifs.tuwien.ac.at/files/MRT2010_tr.pdf
10. Melik-Merkumians, M., Wenger, M., Hametner, R., Zoitl, A.: Increasing portability and reuseability of distributed control programs by i/o access abstraction. In: 15th IEEE International Conference on Emerging Techonologies and Factory Automation - Work in Progress Track. p. accepted for publication (2010)
11. Melik-Merkumians, M., Zoitl, A., Moser, T.: Ontology-based fault diagnosis for industrial control applications. In: 15th IEEE International Conference on Emerging Techonologies and Factory Automation - Work in Progress Track. p. accepted for publication (2010)
12. Merdan, M.: Knowledge-based Multi-Agent Architecture Applied in the Assembly Domain. Ph.D. thesis, Vienna University of Technology (2009)
13. Obitko, M., Marík, V.: Ontologies for Multi-Agent Systems in Manufacturing Domain. In: DEXA '02: Proceedings of the 13th International Workshop on Database and Expert Systems Applications. pp. 597–602. IEEE Computer Society, Washington, DC, USA (2002)
14. Oreizy, P., Medvidovic, N., Taylor, R.N.: Architecture-based runtime software evolution. In: International Conference on Software Engineerings (ICSE 2008). pp. 177–187. IEEE Computer Society (2008)
15. Tim Berners-Lee, J.H., Lassita, O.: The semantic web. Scientific American 284, 34–43 (2001)
16. Vlter, M., Stahl, T.: Model-driven Software Development. John Wiley (2006)
17. Wuttke, J.: Runtime failure detection. In: Companion of the 30th International Conference on Software Engineering. pp. 987–990. ACM, Leipzig, Germany (2008), 1370219 987-990