Preprints of the 18th IFAC World Congress
Milano (Italy) August 28 - September 2, 2011

Model-based Knowledge Extraction for Automated Monitoring and Control

Christoph Legat * **Jörg Neidig** ** **Mikhail Roshchin** *

* *Siemens AG, Corporate Technologies, Munich, Germany*
(e-mail: {christoph.legat.ext | mikhail.roshchin}@siemens.com)
** *Siemens AG, Industry Sector, Nuremberg, Germany*
(e-mail: joerg.neidig@siemens.com)

Abstract: Typically, Plant Lifecycle Management Systems (PLMS) provide rich functionality for universal asset management and engineering during a design phase of production systems. Completing actual realization of these production systems and bringing them into operational mode turns out that necessary information from a PLMS, provided already during engineering step, will not be coupled with an appropriate system any more. It remains so as well, even when it is necessary to call back specific engineering background information for some scenario (e.g. for automated monitoring and control). Our approach presented here aims in finding an effective solution for this issue comprising the following: (1) a formal logic-based model for flexible information acquisition from a PLMS, and (2) an automated reasoning mechanism, which can be flexibly adopted for an implementation of various applications of the operational mode (e.g. diagnostic functionality). To evaluate our proposed concepts and techniques, we focus on the implementation example using the Siemens PLMS product COMOS.

Keywords: Knowledge Acquisition, Knowledge-based Systems, Plant Lifecycle Management, Diagnosis, Systems Engineering

1. INTRODUCTION

The main purpose of a PLMS is the coordination and integration of various trades involved in the engineering process for a future production system (e.g. automation design, electrical wiring and plumbing, asset and maintenance management). Therefore, some common data model is always provided, allowing to plan and model a production system, according originally to its functions, rather than its trade-specific structure.

Although todays PLMS are powerful tools, the major flaw of current approaches is the strong distinction between design and operation phases of the plant lifecycle. In other words there is an insufficient information flow between the engineering and operation tasks of a plant. In general, this leads to an inefficient duplication of work. To bridge this gap, it is required to couple a real automation system efficiently with its virtual counterpart of the digital engineering. Especially cross-functional tasks as automated monitoring and control applications could benefit greatly from such an integrated approach.

In this paper we describe our approach to couple both facing worlds by providing techniques for model-based information extraction from a PLMS and further automated processing with logic-based reasoning techniques for various applications of automated monitoring and control. First of all in Section 2 we analyze and summarize relevant proposals, dealing already with related issues. Existing PLMSs and possible ways of their extensions are presented in Section 3. In order to provide a generic approach and to prove its applicability and feasibility for most PLMSs, we focus our idea on a common data model of a PLMS in Section 4, which provides a basis and sufficient means for the necessary enhancement of a PLMS playing a central role in this paper. Section 5 gives an example, of how the proposed approach can be realized for the diagnosis of a specific automation component.

2. STATE OF THE ART

Some efforts to bridge engineering phases with production system operations has been already done on the base of XML-based data exchange formats, such as CAEX (IEC (2006)) and AutomationML (Drath et al. (2008)). Schleipen et al. (2008) presents an approach for automatic configuration of a monitoring application whereas Schmidberger and Fay (2007) extracts specific rules for Asset Management. These concepts present application-specific mappings in contrast to our generic application-independent information extraction approach. The major disadvantage of CAEX-based approaches is that XML follows a hierarchical model structure, which is quite restricted, compared to a relational model. It only defines a static view of an actual information. So, to improve performances in sharing and discovering information, i.e. to implement an intelligent system, there is a need to add value in the field of relationships between objects. First approaches to overcome these drawbacks has been developed e.g. in Runde et al. (2009).

The integration of engineering knowledge using its inherent semantics is addressed e.g. in Brecher et al. (2010); Wiesner et al. (2008). They provide promising approaches using a common information pool with formal seman-

tics. Nevertheless, it is neither provided a generic concept for knowledge extraction nor answered how to apply the knowledge beyond the engineering phase.

A number of diagnostics models applicable for a wide area of functionalities has been proposed as e.g. in Struss and Ertl (2009) for a component-oriented and consistency-based diagnosis. For a comprehensive overviews on model-based diagnostic techniques see e.g. Isermann (2005); Ligeza (2004). Each of these models can be used as a target model for our system. We apply an earlier developed diagnosis model presented in Fischer et al. (2009) to demonstrate our extraction scenario.

One of the leading PLM softwares is the Siemens product COMOS. Its strength is the fully integrated object orientation, i.e. the description of a plant is completely viewed from the component's perspective. All the different aspects of a component (automation, electronics, etc.) are just different object properties. This architecture eases the integration of additional aspects in a built-in manner. Thus, the COMOS was chosen for the presentation of the diagnostic modeling approach. An additional advantage of the integrated engineering is the straight-forward way to implement the reactions to the diagnostic results, e.g. visualization, event logging, or even automated scheduling of maintenance tasks.

3. EXTENDING PLMS WITH A MODEL-BASED APPROACH

Multiple fast-changing factors, such as technology, product designs and market demands, affect automated production as stated in Westkämper (2003); Wiendahl et al. (2007). Thereby, the following criteria are strongly required and have to be taken into account, choosing solutions for the issue discussed here: (1) flexibility with respect to possible modifications of operations as, for instance, in case of automated monitoring and control scenarios, (2) extensibility of existing models with new properties as, for instance, evidence measures for failure detection or confidence factor for fault isolation tasks; and (3) low-cost realization of new operational features (e.g. new functions for automated maintenance adaptation). Consequently, a major requirement for a flexible enhancement of a PLMS is to provide a generic solution, which allows low-effort adaptability in case of changes and modifications.

From the architectural point of view, there are two different alternatives to realize such kind of an extension for a PLMS: either by conventional classical programming or knowledge-based programming, which comes together with automated reasoning. As discussed in Fischer et al. (2009), classical programming offers the possibility to hard-code knowledge about required functionality (e.g. diagnostics, maintenance) explicitly and symbolically within the structure of the software code. For our idea of a possible extension of a PLMS, it would mean the following: (a) each specialized extension is always tailored to a specific functionality; (b) knowledge about required functionality and additional constraints are usually implicit in the structure of the program; (c) thereby, this knowledge has to be hard-coded by a software engineer, and (d) if there are some changes in a PLMS or operational mode of an appropriate automation system, then the extension has to be revised and modified. This obviously hinders extensibility and flexibility of such approaches, where changes and modifications happen regularly. This is why our concept is based on the idea of knowledge-based programming, which is similar to the idea of the adaptable knowledge-based factories, identified by Jovane et al. (2009) as a strategic future target for competitive and sustainable manufacturing.

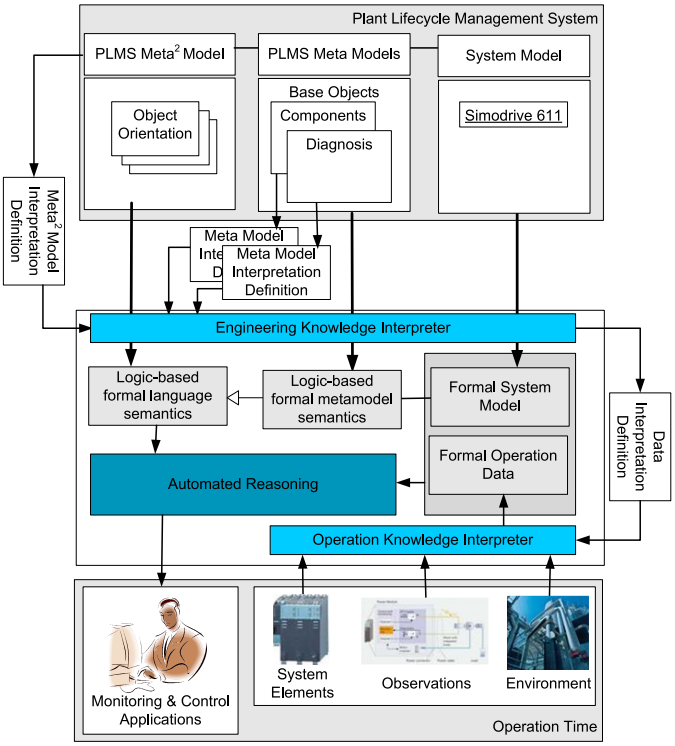

Fig. 1. Overview of the proposed logic-based architecture.

Knowledge-based programming principles provide: (a) automated reasoning mechanisms for generic problem solving, which is independent of a specific functionality; (b) knowledge about required functionality and additional constraints are explicitly represented in a formal logic-based model, using so called description logics (see Baader et al. (2003) for a comprehensive introduction); (c) necessary knowledge is automatically acquired from a PLM system, independently from the automated reasoning mechanism or some software code; and (d) if there are some changes of any parameters, formats or data of a production system, only the logic-based model has to be modified, the automated reasoning mechanism remains the same. Thus, the central role of the approach relies on the logic-based model (discussed in detail the Section 4) and the appropriate automated reasoning (see Sect. 5.2) where necessary engineering information is utilized either when needed or once, when the design process is completed. An overview of our approach to extend a PLMS is shown in Fig. 1, its components are introduced in the following.

Plant Lifecycle Management System: It provides sufficient means for engineering and design of production systems. Such a system design after its completion is called **System Model**. A System Model is a specific realization of various views on a production system designed within a PLMS. These views are generic enough to provide common

information templates for the system engineering procedure also known as **Meta Models** of a PLMS. Some often required meta models are specifically preassigned for certain engineering domains or functionality. For instance, a "Component View" enables to model part-whole relationships between components, and an "Automation View" gives specific definitions of templates for automation control in a future production system. In order to ensure the interoperability among various views of a PLMS and their mutual use during engineering, a PLMS offers a common data model, also known as **PLMS Meta^2 Model (M^{2}M)** defining a basic language with inherent language features. They enable to model various aspects of a future production system, as e.g. trade-specific or functional-oriented. Consequently, a PLMS offers a three-layered data model.

Meta^2 Model Interpretation Definition: First of all, it is required to define a consistent translation and mapping of a PLMS common data model (i.e. M^{2}M) into appropriate the logic-based formal specification (i.e. description logics). It can be realized in different ways, e.g. as XSLT-based transformation whether a XML format of a PLMS M^{2}M is available. In order to keep the generalization of our approach, we apply a model-driven transformation based on rules. The body of such a rule specifies the PLMS M^{2}M language constructs whereas the head of that rule are appropriate DL operators.

Meta Model Interpretation Definitions: Company and task specific views require custom transformations. Therefore, further rule-based interpretations how to apply knowledge in operations is provided. Based on both interpretations (M^{2}M and Meta Model) a system model will be acquired automatically and consistently.

Engineering Knowledge Interpreter: It performs the interpretation of the engineering knowledge given in a PLMS and provides it to applications to applications during operational phase (e.g. for automated monitoring and control). Accordingly, the system model will be automatically analyzed, based of the previous interpretation definitions for M^{2}M and Meta Model.

Operation Time: During operational phases of an automation system, monitoring and control applications enable an early detection of some automation faults and failures, based on data and observations from an appropriate system. In this paper it is assumed that most tasks of automated monitoring and control can be provided with the help of automated reasoning, discussed later.

Data Interpretation Definition : Independently on a data format applied by a production system, engineering knowledge e.g. about system components used in the field, is required to achieve a correct interpretation of operation data. This is encapsulated through the Data Interpretation Definition.

Operation Knowledge Interpreter: A production system operates information in a number of different formats, from standardized ways as e.g. OPC UA or IOLink to proprietary protocols. The interpretation of the operation data is out of scope of this paper and left to future developments. Based on the formal model definitions generated by the Engineering Knowledge Interpreter, operation data is interpreted by the Operation Knowledge Interpreter to offer an adequate formal knowledge base for automated reasoning.

Logic-based Formal Language Semantics: It enables automated processing and reasoning of engineering information with operation data during run time of a production system. A formal language definition of the PLMS common data model (M^{2}M) is necessary in order to ensure correct and consistent engineering information reuse, where M^{2}M encapsulates all model inherent features of an appropriate PLMS. The reason to chose a logic-based approach is that most necessary functionalities of monitoring and control applications can be formally defined as logic-based reasoning tasks, and are therefore solved through the application of generic reasoning algorithms, specific for the chosen logic family. The logic-based model is a formal representation of both syntax and semantics of a M^{2}M, and it provides even higher expressivity than a common data model itself. Section 4 describes the logic formalism applied in our approach.

Logic-based Formal Meta Model Semantics: : It contains the interpretation of elements of PLMS views whether predefined by the PLMS provider or the customer itself. It is generated automatically by the Engineering Knowledge Interpreter with the help of specific interpretation definitions (if needed) to provide an adequate model base for a specific operation functionality.

Formal System Model: It defines a particular realization of a PLMS Meta Model. It is also generated automatically by the Knowledge Engineering Interpreter.

Formal Operation Data: These data are logic facts generated by the Knowledge Operation Interpreter. They represent a current situation of a particular production system and its environment.

4. LOGIC-BASED MODELING FOR AUTOMATED KNOWLEDGE EXTRACTION FROM PLMS

In our approach we propose to use a logic-based model, which is an appropriate reflection of the information structure used within a PLMS. Thus, it remains consistent for further automated model-based knowledge extraction procedures and also automated monitoring and control functionality. Consequently, a transformation of the PLMS M^{2}M into a corresponding logic-based model is required. Once it is provided, the automated knowledge extraction mechanism can be performed.

The selection of an appropriate language for serialization involves various aspects such as expressivity (availability of language constructs), query complexity/performance, and the availability of standards. Here we rely on languages with formal semantics, i. e. languages, whose constructs are defined with formal logics. Our suggestion is to use the Web Ontology Language (OWL) whose semantics is based on description logics providing some favorable properties for our application:

(1) OWL comes with a syntax and semantics standardized by the W3C (see Motik et al. (2009b) for details). The standardization facilitates interoperability in cross-company settings and enables a solid tool chain with commercial as well as open source products.
(2) Production systems rely to a great extend of hierarchical models. Automated concept classification and consistency checking facilitates the description of necessary reasoning tasks for automated monitoring and control applications.
(3) In order to support efficient queries, OWL 2 provides tractable fragments with restricted expressivity. For our purposes a fragment such as OWL 2 EL provides both sufficient expressivity and only polynomial complexity.
(4) Since information is collected from various sources during operation time, another important property of description logics is the declarativity of the language. Thus, the sequence knowledge is added to the knowledge base does not influence the querying results.

A comprehensive introduction to OWL is beyond the scope of this paper. The interested reader is referred to Hitzler et al. (2009).

4.1 COMOS Meta^2 Model

In this paper, we consider COMOS as an exemplary PLMS, where the common data model (PLMS M^{2}M) is based on the object-orientation principles. The major advantage of objects in the context of COMOS is a native support for inheritance, which directly implies hierarchical relations among objects. Therefore, each object operates with: (i) properties, describing static characteristics of object states, which can either be simple data types or other objects (associated or aggregated), (ii) methods, determining a dynamic behavior of an object and possible influence on its properties, and (iii) events, which are actually messages, defined for possible interface to external application. Object-orientation facilitates also the design of predefined templates (Meta Models), related to some functionality or specific applications, as for instance maintenance, automation, or mechanical constraints. Therefore, each Meta Model is first of all defined within COMOS using such templates - also called base objects or known as classes of objects with predefined properties, methods, and events, specific for a respective environment. When it comes to design of some production system, each recently added object is a subclass of some base object, which is in-turn what we call as a system model or one of its elements.

4.2 COMOS Meta^2 Model Interpretation Definition

In order to achieve a consistent interpretation of knowledge modeled in a PLMS, interpretation definitions encapsulate the rules how to transform a PLMS Meta^2 Model into the logic-based formalism. We apply a rule based notion for the interpretation definition. The body of a rule consists of a notion describing the pattern within the PLMS M^{2}M (which can be realized by PLMS specific build-ins) whereas the head of the rule defines the structure of the description logics representation. As already mentioned, the COMOS M^{2}M is based on object-orientation which can be mapped without expressivity loss to description logics as described in Gaaevic et al. (2006). An extract of the interpretation definitions used by the Engineering Knowledge Interpreter to transform the COMOS Meta^2 Model to description logics language is given in Table 1.

Table 1. Interpretation definition of the COMOS Meta^2 Model

PLMS-specific rule body	$\rightarrow$ DL-specific rule head
`Object`(o)	`Concept`(o)
`Object`(o_1) $\wedge$ `Object`(o_2) $\wedge$ `inherit`(o_1,o_2)	`Concept`(o_1) $\wedge$ `Concept`(o_2) $\wedge$ `OWL:subclassOf`(o_1,o_2)
`Object`(o_1) $\wedge$ `Object`(o_2) $\wedge$ `COMOS:associatedTo`(o_1,o_2)	`Concept`(o_1) $\wedge$ `Concept`(o_2) $\wedge$ `OWL:assocTo`(o_1,o_2)
`Object`(o) $\wedge$ `Attribute`(a) $\wedge$ `COMOS:hasAttribute`(o,a) $\wedge$ `COMOS:hasDatatype`(a,d) $\wedge$ `COMOS:hasValue`(a,v)	`Concept`(o) $\wedge$ `Concept`(a) $\wedge$ `OWL:hasAttribute`(o,a) $\wedge$ `OWL:hasValue`(a,d) $\wedge$ `OWL:hasDatatype`(a,v)

4.3 Integration of Meta Models in COMOS

In order to integrate diagnostic functionality into the PLMS, which provides engineers with sufficient means for further automated monitoring and control functionality of a respected production system and also common understanding among various trades about possible faults and failures, detection and isolation mechanisms, a specific PLMS Meta Model has to be integrated. One of the most general description of diagnostic terminology can be obtained from the ISO 13379 (see ISO (2003)). In terms of this standard there are two major base objects to be included into the Diagnosis Meta Model: Symptom and Diagnosis, responsible for fault detection and isolation, respectively. Additionally, it is constrained that each diagnosis can consist of symptoms as given in the UML notion of the Diagnosis Meta Model in Figure 2.

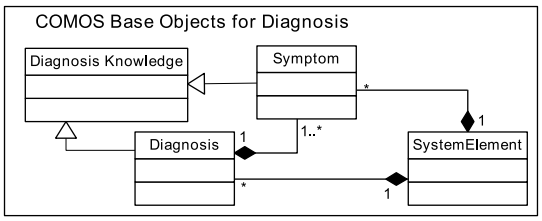

Fig. 2. UML notion of the Diagnosis Meta Model.

4.4 Diagnosis Meta Model Interpretation Definition

Our approach aims in a reasoning-based implementation of diagnosis functionality, based on observations of some compound system. To this effect our logic-based model should be extended with additional semantics without changing already acquired information from a PLMS. This means that the information related to the engineering models of an appropriate production system remains the same, and only some extensions related to the required functionality for automated monitoring and control has to be provided, using additional expressivity means. Functionality specific definitions of corresponding meta models have to be additionally given using Meta Model Interpretation Definitions. Thereto, we apply rules in the same way as described for the M^{2}M Interpretation Definition. Additionally, in the body of the rule, language features of the formal semantics of the PLMS Meta^2 Model can

be used to ease the formulation of definitions as e.g. by taxonomic relationships.

Additional semantics has to be identified for the relations among system components, and between system components and their diagnosis-related objects. It is obvious that the default relations acquired from the COMOS M^{2}M can have different meanings, such as "consistsOf" for inherited components of a system. The same should be done for the associations between components, symptoms and diagnoses, e.g. "hasSymptom" is more specific than "associatedTo".

For the Component View an additional Meta Model Interpretation Definition to introduce the semantic of part-whole relationships is given as follows:

$\texttt{Component}(x) \wedge \texttt{Component}(y) \wedge \text{inherent}(x,y) \rightarrow \text{consistsOf}(x,y)$

In order to rename components, and to be more specific, we call them system elements. The appropriate rule looks as follows:

$\texttt{Component}(o_1) \rightarrow \text{SystemElement}(o_1)$

To incorporate the specific semantics of the Diagnosis Meta Model, where an association of objects of the COMOS Meta^2 Model should imply more specific relationships, additional rules are applied:

$\texttt{Diagnosis}(x) \wedge \texttt{Symptom}(y) \wedge \text{assocTo}(x,y) \rightarrow \text{hasSymptom}(x,y)$

The information acquired from the system model, which addresses the both views: component and diagnostic, where components are related to their possible symptoms and diagnoses, can be transformed through the Meta Model Interpretation as follows:

$\texttt{Component}(x) \wedge \texttt{Symptom}(y) \wedge \text{assocTo}(x,y) \rightarrow \text{hasSymptom}(x,y)$

$\texttt{Component}(x) \wedge \texttt{Diagnosis}(y) \wedge \text{assocTo}(x,y) \rightarrow \text{hasDiagnosis}(x,y)$

After executing the Meta Model Interpretation, the major concepts defined in the rules above are `Symptom`, `Diagnosis` and `SystemElement` together with their respective relationships.

5. APPLICATION EXAMPLE

The transformation of both meta models and system models appropriate to a specific production system is completed automatically, based on the previously described interpretation rules and the execution of the Engineering Knowledge Interpreter. Let us consider a concrete implementation of the approach, using one example for diagnostic functionality.

5.1 Modeling Diagnosis of Simodrive 611

As compound system we take the product Simodrive 611, which is a drive control system, used for different machine tool automation solutions, where combination and further flexible synchronization of complex drives is required. A Component Meta Model for Simodrive 611 consists at least of a three base objects: `Simodrive 611`, `ControlUnit` and `PowerSupply`, as shown in Figure 3 on the right hand side (no cardinality of aggregations denote cardinality one). It is also constrained that each drive system has at least one control unit and one power supply. Additionally the relevant information about possible observations are given in terms of attributes: a control unit has a signal for transformation functions, and a power supply has two LED lamps - Status LED and Charging LED. All these attributes are defined through boolean values.

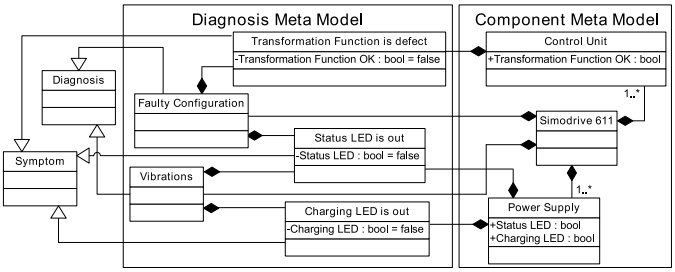

Fig. 3. Meta Models of a Simodrive 611

Diagnosis Meta Model There are two known failures, which can be defined during engineering procedure: faulty configuration and vibrations of the complete system, where a Simodrive 611 performs. Both failures have their own signs, originating from the two subcomponents: control unit and power supply. As discussed previously, in order to use the predefined knowledge about diagnoses for some production system, there should be first of all a diagnostic view used for the system model design. Thus, the Meta Model for diagnosis knowledge presented in Section 4 is used here.

Each detection of a failure is defined as subclass of the base object Symptom, using relevant attributes: transformation function, Status LED, and Charging LED. The attributes use default "false" values (see Figure 3 on the left hand side). Each possible fault definition is a subclass of the base object Diagnosis: `FaultyConfiguration` and `Vibrations` respectively. If the transformation function of a drive control unit is defect, and at the same time, the status LED of a power supply module is out, it can be concluded that the respective Simodrive 611 has a faulty configuration. Additionally if both LEDs of a power supply module are out, a respective Simodrive 611 has high vibrations. Some components are owners of the symptoms, e.g. `ControlUnit` is responsible for the transformation function, and `PowerSupply` consists of both LEDs.

Interpretation of Models Both previously presented meta models (Component and Diagnosis) are interpreted by the Engineering Knowledge Interpreter applying the rules given in Table 1 and Section 4.4. An extract of the actual symptom and diagnosis definitions look as follows (we use a rule-based notion for OWL statements as described in Motik et al. (2009a)):

$\texttt{TransformationFunctionIsDefect}(o_1) \leftarrow \texttt{hasAttribute}(o_1,o_2) \wedge \texttt{TransformationFunctionOK}(o_2) \wedge \texttt{hasValue}(o_2, \textit{false})$

$\texttt{StatusLEDisOut}(o_1) \leftarrow \texttt{hasAttribute}(o_1,o_2) \wedge \texttt{StatusLED}(o_2) \wedge \texttt{hasValue}(o_2, \textit{false})$

$\texttt{FaultyConfiguration}(e) \leftarrow \texttt{hasSymptom}(c,a) \wedge \texttt{hasSymptom}(d,b) \wedge \texttt{StatusLEDisOUT}(a) \wedge \texttt{TransformationFunctionIsDefect}(b) \wedge \texttt{consistsOf}(s,c) \wedge \texttt{consistsOf}(s,d) \wedge \texttt{hasDiagnosis}(s,e)$

5.2 Model-based Diagnosis of Simodrive 611

Now it is possible to directly use operation data for automated diagnosis tasks under the assumption that the operation data is given in an adequate format by the Operation Knowledge Interpreter. The description how to interpret operation data in a correct way is given in the Data Interpretation Definitions which enables e.g. to set the status of LEDs correctly to the corresponding value in the logic-based model either with "true" or "false". All other symptoms are interpreted analogously.

Using the generic reasoning algorithms, the diagnostic functionality is performed automatically without any additional programming, simply executing the description logics inferencing task for logic-based model classification. If the observation about the transformation function provides the value false, then the symptom `TransformationFunctionIsDefect` will appear, but no diagnosis yet. If additionally, the observation `StatusLED` comes with false value, and automatically yields the appropriate symptom `StatusLEDisOut`, then the diagnosis `FaultyConfiguration` will be automatically classified.

It is also possible to provide additional functionality for monitoring and control application as e.g. classifying also hypothetical diagnoses. Semantically, it requires an additional interpretation definition of the respective hypothetical diagnoses (indicated by the prefix "HD") as e.g. `HD_FaultyConfiguration`, where the definition is provided in the same way as previously defined, but instead of using logical operator conjunction, there will be used the disjunction operator.

$$\texttt{HD\_FaultyConfiguration}(e) \leftarrow \texttt{hasSymptom}(c,a) \wedge \texttt{hasSymptom}(d,b) \wedge [\texttt{StatusLEDisOUT}(a) \vee \texttt{TransformationFunctionIsDefect}(b)] \wedge \texttt{consistsOf}(s,c) \wedge \texttt{consistsOf}(s,d) \wedge \texttt{hasDiagnosis}(s,e)$$

Thus, if only one symptom `StatusLEDisOut` is triggered, then both diagnoses of the example depicted in Fig.3) will be automatically classified as hypothetical. In the same way other possible functionality based on logic-based reasoning functionality and model-based knowledge extraction from a PLMS can be provided (e.g. for maintenance).

6. CONCLUSION

In this paper, we have presented a new practical and flexible approach for model-based knowledge extraction from a plant lifecycle management system for monitoring and control of a production system. It brings together both facing worlds: rich functionality for diagnosis knowledge engineering from a PLMS and advanced possibilities of formal logic-based models with automated reasoning, used for easier integration of both operating data and predefined engineering knowledge.

REFERENCES

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P. (2003). *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press.

Brecher, C., Özdemir, D., Feng, J., Herfs, W., Fazullin, K., Hamadou, M., and Müller, A. (2010). Integration of Software Tools with Heterogeneous Data Structures in Production Plant Lifecycles. In *10th IFAC Workshop on Intelligent Manufacturing Systems*, 55–60.

Drath, R., Peschke, J., and Lips, S. (2008). AutomationML - The Glue for Seamless Automation Engineering - Top Level Architecture.

Fischer, J.G., Roshchin, M., Langer, G., and Pirker, M. (2009). Semantic Data Integration and Monitoring in the Railway Domain. In *IEEE IRI 2009*, 11–16.

Gaaevic, D., Djuric, D., Devedzic, V., and Selic, B. (2006). *Model Driven Architecture and Ontology Development*. Springer-Verlag.

Hitzler, P., Sebastian, R., and Krötzsch, M. (2009). *Foundations of Semantic Web Technologies*. Chapman & Hall/CRC, London.

IEC (2006). IEC 62424: Representation of Process Control Engineering - Requests in P&I Diagrams and Data Exchange between P&ID Tools and PCE-CAE Tools.

Isermann, R. (2005). *Fault-Diagnosis Systems: An Introduction from Fault Detection to Fault Tolerance*.

ISO (2003). ISO 13379 : Condition Monitoring and Diagnostics of Machines – General Guidelines on Data Interpretation and Diagnostics Techniques.

Jovane, F., Westkämper, E., and Williams, D. (2009). *The ManuFuture Road - Towards Competitive and Sustainable High-Adding-Value Manufacturing*. Springer.

Ligeza, A. (2004). *Selected Methods of Knowledge Engineering in Systems Diagnosis*, chapter 16, 633–674.

Motik, B., Grau, B.C., Horrocks, I., Wu, Z., Fokoue, A., and Lutz, C. (2009a). OWL 2 Web Ontology Language: Profiles. W3C Recommendation.

Motik, B., Patel-Schneider, P.F., and Parsia, B. (2009b). OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax. W3C Recom.

Runde, S., Güttel, K., and Fay, A. (2009). Transformation von CAEX-Anlagenplanungsdaten in OWL Eine Anwendung von Technologien des Semantic Web. In *Automation 2009*, 175–178.

Schleipen, M., Drath, R., and Sauer, O. (2008). The system-independent data exchange format caex for supporting an automatic configuration of a production monitoring and control system. *IEEE International Symposium on Industrial Electronics*, 1786 – 1791.

Schmidberger, T. and Fay, A. (2007). A Rule Format for Industrial Plant Information Reasoning. *IEEE ETFA 2007*, 360 – 367.

Struss, P. and Ertl, B. (2009). Diagnosis of Botteling Plants - First Success and Challenges. In *20th International Workshop on Principles of Diagnosis*, 83–90.

Westkämper, E. (2003). Adaptable Production Structures. In *Manufacturing Technologies for Machines of the Future (21st Century Technologies)*, chapter 4, 87–120.

Wiendahl, H.P., ElMaraghy, H.A., Nyhuis, P., Zäh, M.F., Wiendahl, H.H., Duffie, N.A., and Brieke, M. (2007). Changeable Manufacturing - Classification, Design and Operation. *Annuals of the CIRP*, 56(2), 783–809.

Wiesner, A., Wiedau, M., Morbach, J., Marquardt, W., Temmen, H., and Redmer, J. (2008). Ontology-based Integration and Management of Distributed Design Data. In *Collaborative and Distributed Chemical Engineering - From Understanding to Substantial Design Process Support*, volume 4970 of *LNCS*, 647–655.