Model-based Diagnosis in Manufacturing [1]

Martin Ilkerl[2] and **Markus Stumptner, Franz Wotawa**[3]

Abstract. Given the increasing use of digital control systems in manufacturing plants, the theoretical potential for error monitoring and diagnosis has significantly increased. However, conventional monitoring systems often still suffer from the problems of past decades where automated error recognition is nonexistent and the monitoring system restricts itself mainly to communications and signal processing. As a result, the diagnostic load in case of malfunctions is left to the operator. We describe a model-based diagnosis system that is integrated into the control of a manufacturing plant by providing an additional layer atop the existing monitoring system. The diagnosis system provides flexible, high-level and resilient diagnosis capability and can directly use the monitoring system's reports as observations for the diagnosis process. We also discuss general principles of developing models for this domain and show a simple model used in our current diagnosis/monitoring prototype system.

1 Introduction

Model-based and qualitative reasoning has been successfully applied to different domains including the automotive domain [5, 1], monitoring of industrial plants [2], and even software debugging [4, 7]. In this paper we describe the use of model-based diagnosis [6, 3] for helping to assist supervision of manufacturing processes. In contrast to previous research, such as [2] we face a situation where the manufacturing plant together with a control and monitoring system had been completed and well defined interfaces to the systems are available. Changing existing control and monitoring systems to introduce model-based concepts is nearly impossible, because those systems are in practical operation and changes are too expensive. On the other hand, improving maintenance operations can result in decreased maintenance costs by reducing downtimes. In this paper we will limit ourselves to a particular type of plant, namely, those dealing with the handling and processing of discrete goods. This includes packaging or assembly lines, but we will not consider dealing with processes such as, e.g., chemical reactions.

The hardware/software architecture of a typical plant is depicted in figure 1. At the bottom level, the plant comprises actuators, sensors, wires for control and power supply, and other hardware, e.g., computers. Based on the hardware level a control level ensures the intended operation. Nowadays, control of the manufacturing process is shifting more and more from the hardware level to software for increased flexibility. At the top level, a monitoring system presents the current state of the plant to the operator. If the state leaves the normal operations envelope, error messages are generated and the operator is informed. At this stage the operator has to schedule actions in order to bring the plant back to the normal state. Although there are manuals mapping error messages to repair actions, the source of the faulty behavior is not always easily detectable, usually because error messages describe only local events, and the general view of the plant and the connections between subsystems are not represented in a traditional monitoring system. Another difficulty is that sometimes a burst of messages caused by a single fault can overwhelm the operator.

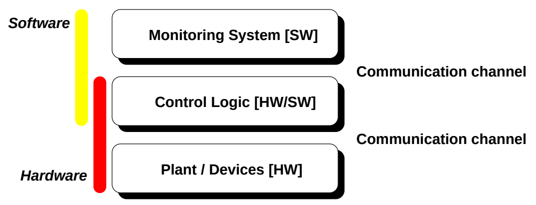

Figure 1. The HW/SW architecture of a plant

Two examples should illustrate the problems with traditional monitoring systems, both obtained from experience with a real plant in the printing industry. First, assume a machine cut objects into pieces to be further processed. Since a cutting device is dulled over time the time, the edges of the pieces lose sharpness as well. This can cause a fetching problem for other machines trying to grasp the piece. In this case, the monitoring system will generate an error message regarding the grasping operation. This message in turn causes the operator to look at a part of the plant not really responsible for the fault. As another example, assume that a subgroup of the devices on the manufacturing floor is connected to the same fuse in the power network. If that fuse blows, the monitoring system will create a message burst as mentioned above concerning the missing or incorrect effects of all machines affected by the fuse. Directly deriving the source of the fault is very difficult since the state of the fuse is not monitored. To overcome these problems, deep knowledge about a factory is obviously required and model-based reasoning provides an optimal framework for representing and using such knowledge.

In this paper we present the work we have done to introduce model-based diagnosis into the monitoring system of a real-world manufacturing plant. The basic idea is to introduce an additional diagnostic system layer on top of the monitoring system (see figure 2). The diagnostic system possesses knowledge about the hardware (and maybe the software) and is intended for filtering unnecessary error messages and guiding the operator in locating the fault. Currently, we have implemented the model-based engine and its interface to the monitoring system, with the full model of the plant being in development. The basic concepts for creating the model from the available data, the design of the diagnosis engine and of the interface are described in the following sections. We demonstrate the modeling pro-

[1] The work described in the paper has been partially supported by FESTO Austria and the Austrian Science Fund Project N Z29-INF.

[2] Festo Austria, Lützowgasse 14, A-1140 Wien, Austria, Email: im.festo@attglobal.net

[3] Technische Universität Wien, Institut für Informationssysteme, Favoritenstraße 9-11, A-1040 Wien, Austria, Email: {mst,wotawa}@dbai.tuwien.ac.at, Authors are listed in alphabetical order

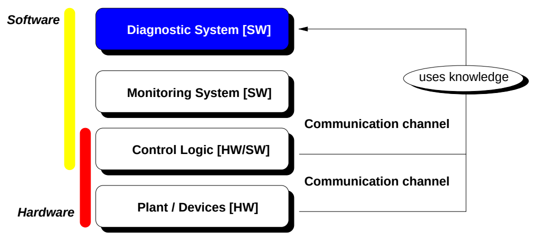

Figure 2. The improved HW/SW architecture

cess using a typical part of a plant. Central to the modeling is the idea of formalizing not just the knowledge itself, but to formulate and eventually formalize modeling principles using available design knowledge. This will enable even inexperienced users to write models for model-based diagnosis, and to provide a sound basis for modeling discipline. Because the modeling principles only require typically available types of knowledge, we expect that they are generally useful.

The paper is organized as follows. In section 2 the typical available design knowledge is described together with the modeling principles that can be derived from it. Section 3 shows the application of the principles using a typical example, followed by a section describing the current implementation and interface to the monitoring system.

2 Available Knowledge and Requirements

Writing a model for a system requires knowledge about the system's structure and behavior, the expected task of the model, e.g., diagnosis, plus possibly other requirements such as minimum expected results. In our domain, device drawings, schematics and board layouts, program listings, lists of error messages, maintenance guidelines, and user documentation of almost all parts of a manufacturing plant are available. Additional knowledge, i.e., informal knowledge, might be obtained from engineers and operators. In the past, knowledge about the electrical and mechanical properties of devices was sufficient to built a model of the correct behavior of the device. Since control functions are no longer implemented in hardware but more and more represented as programs running on a micro-controller or PC, this is no longer true. Therefore, we expect that a flowchart or state diagram representing the (time dependent) behavior of a device is available. In summary, we assume that the following information is available for modeling:

Drawings, Schematics Drawings representing the assembly of mechanical and electrical parts can help to identify diagnosis components and connections, i.e., the structural part of the model.

Flowcharts Documents containing flowcharts or verbal descriptions of the expected behavior are used for specifying the behavioral part of the model. Figure 3 shows a flowchart for the cutting example from the Introduction.

Error messages The error messages and maintenance guidelines are written for the benefit of the operators who, supported by the monitoring system, try to identify possible observations for the diagnosis system. Usually, each entry in the error message list has a unique error number, describes the surroundings of the fault, the involved sensors and their values, and lists posssible causes of the error together with repair actions. See Figure 4 for an example item in such a list.

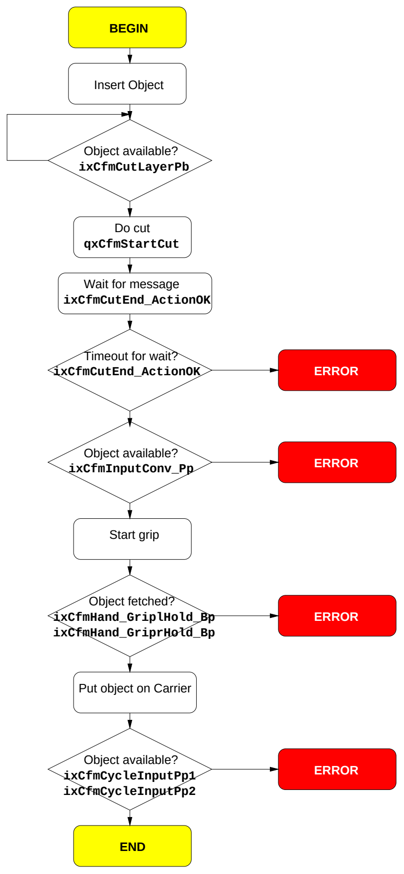

Figure 3. A flowchart for a monitoring system

In comparing the error message from figure 4 with the flowchart from figure 3, we see that the sensor `ixCfmHand_GriplHold_Bp` is used in both. Since the value of the sensor is observed by the monitoring system, it should be used as observable connection in the diagnosis system. Other connections (interchange or transport of information, energy, or objects) may not be directly observable by the monitoring system. In order to tightly couple the monitoring system with the diagnosis system, there should be a direct correspondence between the descriptions of both, i.e., diagnosis components should be those components mentioned by the repair manuals or represented by the user interface. This relation need not be one-to-one. For example, the error message from figure 4 indicates a distinction between sensors, valves, and cylinders. If we use only two components, i.e., a sensor and a grip, the valve and the cylinder are assumed to be part of the grip. Hence, a diagnosis containing the grip means that the valve or cylinder or an obstacle causes the faulty behavior. It should be noted that is not necessarily a drawback of the representation that a distinction between all different components is not possible in the example as this may be a property of the actual system. The operator may be unable to identify the real fault using only the monitor-

```
ERR6.2 exCfmHand_GriplHold_Wp -- Left Grip
```
Location
Carrier. A mechanical finger has to ensure that the objects are fixed before lifting and moving them.
Sensors
`ixCfmHand_GriplHoldBp`
Causes
1. Faulty sensor
2. Faulty valve/cylinder
3. Obstacle prevents from reaching the final position

Repair
1. Check sensor and replace
2. Check valve/cylinder and replace
3. Remove obstacle to ensure that the final position can be reached

Figure 4. An example error message

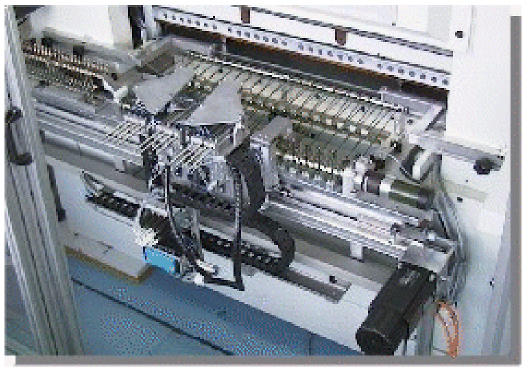

Figure 5. Transport/grip/machine feed combination in a manufacturing plant

ing/diagnosis system and may still need to inspect the device. The following rules are designed to be used for creating a model out of the available information of a manufacturing plant.

- From the line drawings and schematics try to identify diagnosis components, e.g., sensors, actuators. Reduce the number of diagnosis components using the error messages and suggested repair actions. Try to pack components into a single (hierarchical) component.
- Variables, e.g., sensor values, used in the monitoring system are mapped to connections. Additional connections are given by considering objects moved from one device to another, information flow, and power supplies.
- Identify connectivity between components using the suggested connections. Additional components or connections may be necessary.
- Use the flowcharts and additional domain knowledge for creating the behavior part of the model. This may again require additional connections and components.
- After building the structural and behavioral part of the model, attempt to streamline it by considering the following checklist:
  - Do previously designed component models (from other or related domains) exist that can be reused?
  - Can different components be subsumed by one component?
  - Is all relevant knowledge represented by the model?
  - Are additional components/connections necessary in order to get better results, e.g., reduce the number of diagnosis candidates to feasible size?

In the next section we highlight the modeling process using a small real-world example.

3 Modeling

The following model represents a small part of a manufacturing plant (see figure 5). This part comprises a cut, a grip, and a carrier. The cutter cuts up an object into equal pieces. The pieces are grasped by the grip, lifted up and placed on a carrier for further processing. The flowchart of this process is depicted in figure 3. An example of a message created by the monitoring system in case of an error is given in figure 4. In this example, several sensors for detecting the product and states of the actor are involved. Note that the cut operation itself involves several components including a belt for transporting the object to the cutting knife.

Following our rules for creating a model, we first have to identify components. From the technical drawings, flowcharts, and error messages, we can identify several sensors, the cutter, the grip, and the carrier.

In the second step we have to introduce connections. The connections represent sensor values, as well as product, control, and energy flows. The control flow is given by the flowchart. For example the action `qxCfmStartCut` can be mapped to a connection and represents control information.

In the third step we have to build the structural part of the model by formalizing the interconnectivity. In our case we see that we have connections, e.g., `qxCfmStartCut`, which should be driven by a control component. Therefore, we introduce such a component that is connected with the cutter. Note that the control component need not be the representation of a device. Instead it may represent a conglomerate of hardware as well as some parts of the control software. The structural part of the final model (ignoring the power supply) is given in figure 6.

Next we have to build the behavioral part of the model. For the model in this paper we make the assumption that time is not important and therefore can be ignored. Faults caused by temporally incorrect behavior are not considered. Note that this assumption is not too strong for our purposes. Timing errors such as transport timeouts are directly detected by the monitoring system and lead to a ”predigested” observation for diagnosis, e.g., no product at the output of the cutter.

Sensors All considered sensors are designed for directly or indirectly observing whether a product is on the desired position or not.

$$\neg AB(X) \Rightarrow (in(X) = product \leftrightarrow out(X) = true)$$

Cut The cutter communicates with a control component. A workpiece is cut after the startCut message has been received. In this case the actionOK message is sent.

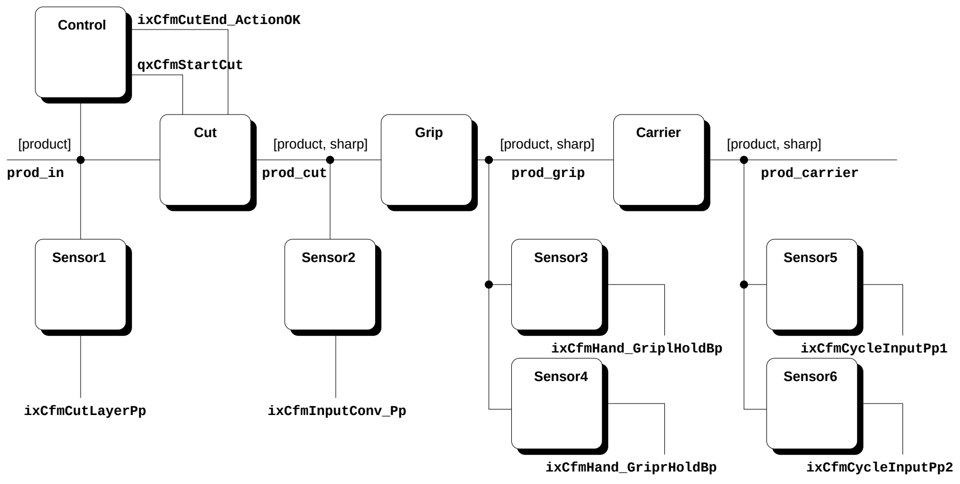

Figure 6. A model for the cut-grip system

$$\begin{aligned}&\neg AB(X) \Rightarrow\\ &\quad (in(X) = product \wedge startCut(X) = true \leftrightarrow\\ &\quad out(X) = product \wedge actionOK(X) = true)\end{aligned}$$

Control The control component checks whether a product is detected or not. If the product is detected, the startCut message is sent.

$$\begin{aligned}&\neg AB(X) \Rightarrow\\ &\quad (in(X) = product \leftrightarrow startCut(X) = true)\end{aligned}$$

Grip The grip takes the product and puts it on the carrier.

$$\neg AB(X) \Rightarrow (in(X) = out(X))$$

Carrier The carrier takes the product.

$$\neg AB(X) \Rightarrow (in(X) = out(X))$$

Note that this is only a partial model of the actual system. Fault models and additional connections, e.g., for power supply, are omitted here for brevity.

We now show how the model is applied when the error message `ERR6.2` from figure 4 is produced by the monitoring system. In this case we get the following observations:

$$\begin{gathered}\texttt{ixCfmCutLayerPp} = true\\ \texttt{ixCfmInputConv\_Pp} = true\\ \texttt{ixCfmHand\_GriplHoldBp} = true\end{gathered}$$

From the first two facts and the model, we derive $\texttt{prod\_in} = product$ and $\texttt{prod\_cut} = product$. Assuming that **Grip** is correct leads to the conclusion $\texttt{prod\_grip} = product$ and only the assumption that **Sensor3** is faulty ensures consistency. Hence, one diagnosis is that the sensor may be faulty. The second diagnosis, i.e., that **Grip** is faulty, can be derived by assuming that the sensor is correct. A model-based diagnosis engine as described in [6] would return these two single diagnoses. What we see is that the model allows to derive the same information as given by the error message description in the repair manual.

The difference of the model-based approach to repair manuals is that it is neither restricted to single faults nor potentially incomplete. Instead, the model can be used in unexpected situations where multiple faults are possible, and the results will be complete with regard to the model. In addition, model fragments can be reused in other models, and model changes for specific components do not have an effect on other component models. Finally, while repair manuals mostly describe local cause-and-effect relationships, the model-based approach does not suffer from this limitation. In reality, it is quite possible that faults occurring at a previous stage of plant operation are detected in a later stage of the process. Therefore, for diagnosis it is necessary not only to see a part of a process but the whole. This is ensured by the model-based approach.

4 Implementation

The implementation of the prototype comprises several layers. The structured Layers are depicted in figure 7.

Java VM: The underlying implementation platform.

Diagnosis System Kernel: The monitoring and diagnosis algorithm.

JDBC Interface: The database interface. Actually the system is designed to work as an open system, therefore file system and database operations are supported. This layer is used to load model and system definitions as well, as well as store history data for evaluation purposes.

Native Interface: To allow integration with existing systems or software packages (e.g. visualization programs), this layer is responsible for low level data exchange, possible programming language specific (e.g. C++ interface).

Definition Interface: The JDBC interface is the low-level access to the database. This is encapsulated via the definition interface, so that it is possible to implement e.g. file-access for low-cost versions.

Online Interface: This interface is responsible for receiving data from foreign systems and to add those values as observations to the diagnosis system. A parser is integrated to translate data from the process to the internal representation.

Processes: All foreign systems which are to be integrated with the diagnosis system are described with this layer. As described in the native interface, it is possible to communicate with Java-based systems and with systems in general that have C and C++ interfaces.

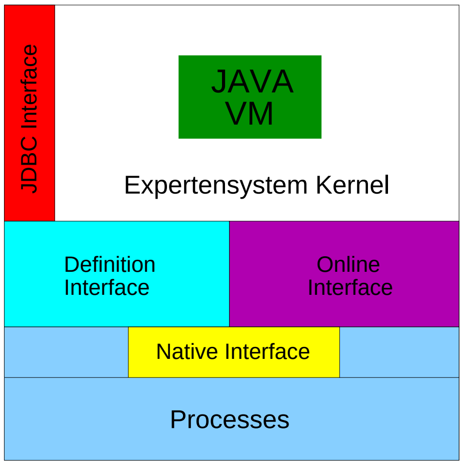

Figure 7. The structured layers of the prototype

4.1 Software Structure

The monitoring system, which interfaces between user and process, is responsible for the communication between the automation part and the diagnosis kernel, so this is one of the set of processes as described above.

The system was designed either to use existing visualization software that sends and receives the necessary data (e.g. object values from the automation plant) or to implement a driver that directly communicates with the plant. Therefore an special interface `CForeignLowLevelInterface` is responsible for getting and setting values [functions: `readForeignSystemValue(CForeignSystemObject CFObject)`, `setForeignSystemValue(CForeignSystemObject CFObject)`]. Each object that represents values from a so called Foreign System has to correspond to the interface specification to allow automatic translation of values into the internal diagnosis system representation.

A Foreign-Object-Definition looks like this:

```
CFObject(NAME, OBJECT TYPE,OBJECT ID,
    FORMULA, DESCRIPTION)
```

`NAME`:	Object name that is known from the Expert system-Kernel.
`OBJECT TYPE`:	Type of the object: • Connection: To receive values from the foreign system. • Component: An object type that is just used to inform this object when an abnormal state was found.
`OBJECT ID`:	This is the ID that identifies a foreign system object.
`FORMULA`:	Formula to filter a value from the foreign system to get the value of an object, e.g. bit masking.
`DESCRIPTION`:	Used to give a textual description of the object.

The monitoring system has the possibility to work either in polling mode or in a call-back mode, depending on the foreign system. If the polling mode is active, the monitoring system cyclically checks a well defined object from the foreign system, whose states is the trigger in case an error occurs. In call-back mode, the foreign system is responsible for actively triggering the monitoring system. Independent of the mode in which the system is triggered, it will read all object values from the foreign system that are related to the error and therefore have to be retained for analysis (”frozen”). These so-called observations are then translated to the diagnosis system representation. Foreign-System-Values are passed on to the diagnosis kernel via the function `addObservation` as described above. The return value is a vector of diagnoses (currently restricted to single errors).

5 Conclusion

Traditionally, error diagnosis in monitoring systems for industrial plants is based on examining abnormal system states in terms of local cause and effect relationships. As a result, much of the interpretative task of expressing faults in plant operation is left to operators who will be presented with low-level error messages describing state deviations noted by individual sensors. Ironically, this means, especially when serious faults occur, operators can be overwhelmed by the amounts of unprocessed information they are presented with, as their displays are flooded with sudden bursts of messages. In addition, although manuals may point out possible and likely types of faults that can produce particular messages, it is left to the operator to integrate the proposals for multiple messages, as well as the actions required to identify, compensate, or repair the fault. The basic properties of model-based diagnosis methods provide an excellent fit to enhance the capabilities of monitoring systems by providing higher-level analysis. Model-based reasoning is founded on the idea of using high-level descriptions of plant operations as a basis for diagnosis. This enables us to combine local knowledge to globally determine sources of faults, describe plants as combinations of components taken from a basic model library, and using generic diagnosis algorithms to diagnose multiple faults for a given plant, based on the component-oriented diagnosis model.

This paper has described the basics of using model-based diagnosis as a support tool for plant monitoring so as to reap the benefits described above. First, we have examined the typically available information from which model-based descriptions of a plant’s functioning can be derived. A central issue here is to make as much use as possible of the existing documentation that was written to support conventional monitoring operations. Based on this description, we have mapped a simple set of principles that describe how the components and connections can be derived from different aspects of the descriptions. We consider this the beginnings of an examination not simply of the model itself, but of analysis of the modeling process.

Next, we showed a simple example model, covering a small part of an actual plant. Despite its abstraction and limited expressiveness (e.g., absence of complex temporal reasoning), this model is able to provide meaningful diagnosis output when applied to actual faults in that part of the plant. Finally, we have described the architecture of the monitoring system prototype, which will allow straightforward integration of the model-based system with the existing monitoring system implementation, allowing for direct access to the monitoring system's results and their use by the diagnosis system.

REFERENCES

[1] Fulvio Cascio, Luca Console, Marcella Guagliumi, Massimo Osella, Andrea Panati, Sara Sottano, and Daniele Theseider Dupré, 'Generating onboard diagnostics of dynamic automotive systems based on qualitative models', *AI Communications*, **12**(1/2), (1999).

[2] S. Cauvin, M.-O. Cordier, C. Dousson, P. Laborie, F. Levy, J. Montmain, M. Porcheron, I. Servet, and L. Trave-Massuyes, 'Monitoring and alarm interpretation in industrial environments', *AI Communications*, **11**(3,4), (1998).

[3] Johan de Kleer and Brian C. Williams, 'Diagnosing multiple faults', *Artificial Intelligence*, **32**(1), 97–130, (1987).

[4] Gerhard Friedrich, Markus Stumptner, and Franz Wotawa, 'Model-based diagnosis of hardware designs', *Artificial Intelligence*, **111**(2), 3–39, (July 1999).

[5] A. Malik, P. Struss, and M. Sachenbacher, 'Case studies in model-based diagnosis and fault analysis of car-subsystems', in *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, (1996).

[6] Raymond Reiter, 'A theory of diagnosis from first principles', *Artificial Intelligence*, **32**(1), 57–95, (1987).

[7] Markus Stumptner and Franz Wotawa, 'Debugging Functional Programs', in *Proceedings* 16^{th} *International Joint Conf. on Artificial Intelligence*, Stockholm, Sweden, (August 1999).