A PRACTICAL METHOD FOR CREATING PLANT DIAGNOSTICS APPLICATIONS

Gabor Karsai, Samir Padalkar, Hubertus Franke[1], Janos Sztipanovits
Department of Electrical and Computer Engineering,
Vanderbilt University, Nashville, TN, 37235

Frank DeCaria
E.I. DuPont DeNemours, Old Hickory Works,
Old Hickory, TN, 37138

Corresponding author:
G.Karsai
P.O. Box 1824-B
Vanderbilt University
Nashville, TN 37235
Phone: (615) 322-2338, Fax: (615) 343-6702
E-mail: gabor@vuse.vanderbilt.edu

[1] Dr H. Franke is with IBM T.J.Watson Research Center currently

Abstract

Making manufacturing systems more robust (i.e. able to carry out their function under the presence of faults) is an issue of paramount importance: it involves on-line and real-time detection of faults, diagnosis of faults, and recovery from the faults. In this paper, we present a system that is able to generate practical and efficient solutions for these problem. This approach we present is available as part of IPCS (Intelligent Process Control System), which is a model-based environment for generating monitoring, control, simulation, and diagnostics applications for large-scale, continuous process plants. IPCS has been used to generate practical real-time diagnostic and recovery applications in chemical and co-generator plants.

Keywords:
plant monitoring and control, fault diagnostics, plant-operations, model-based programming, programming environments, fault recovery

INTRODUCTION

Many manufacturing plants operate under tight budget constraints and have to satisfy increasing production and quality demands. Faults and subsequent lost uptime negatively impact the bottom line; hence a system that can detect, diagnose and recover from faults becomes vital to the success of the company. In this paper, we present a practical ***approach*** and ***programming environment*** aimed at generating diagnosis and recovery applications for large-scale plants.

The generation of diagnosis and recovery applications raises many interesting and complex problems. Fault detection, fault diagnosis, and recovery from diagnosed faults so as maintain production schedules are the problems that we think of at first glance. A common requirement for solving these problems is that they be solved in real-time, i.e. within acceptable time-limits. Another aspect of these solutions is that they must be based on observations of a working, dynamic system: the plant. This brings up all the problems associated with on-line systems, such as nonmonotonicity, continuous operation, asynchronous events, interface to external environments, uncertain or missing data, high performance requirements, temporal reasoning needs, and guaranteed response time [13].

Fault detection involves developing models of normal plant behavior and establishing criteria for violations. An issue that is common to fault detection and fault diagnosis is the optimal allocation and placement of sensors to monitor plant behavior. ***Fault diagnosis*** raises problems of single and multiple faults, propagated faults, sensor failures, feedback loops, different phases of plant operation, active and passive operation, time/resolution trade-offs, and forewarning [19]. ***Fault recovery*** involves issues such as planning, restructuring monitoring and control, and re-scheduling operations.

Operators try to handle these issues on a heuristic level. In many situations their practical, experience-based knowledge of process behavior is sufficient. Plant engineers attempt to tackle these problems by building multiple models: models of plant behavior, models of plant monitoring, models of faults in plants, and models of recovery strategies. These models may reflect a deeper understanding of the process, and, hopefully, may contribute to a better operation of the plant than a heuristic approach. It is the central assumption of this paper that ***models of plant behavior can and should be effectively used in day-to-day plant operations***. Modeling raises the issues of first specifying complex models and later maintaining (i.e. testing and validating, and constantly improving) those models. Since much time is spent in specifying and maintaining these models, it makes sense to make these models as generic as possible so that they can be reused in another application for a similar plant. For a method to be practical for usage in large-scale plants, it obviously has to address all these problems.

This paper is organized as follows. First, a practical programming environment which addresses the problems mentioned above is presented. Next, the fault modeling and diagnostics approach is described, followed by a description of possible fault recovery strategies. After this, a simple application is presented which has been used in a real situation.

IPCS

The approach described in this paper is available as part of IPCS (Intelligent Process Control System) [19] [11], which is a domain-oriented, model-based ***programming environment*** for developing sophisticated monitoring, control, simulation, diagnostics and recovery applications.

The goal of IPCS is to provide an easy-to-use environment which can be employed by engineers with little or no software engineering experience. IPCS achieves this through the use of various ***models***: the models are specified by domain engineers, and IPCS creates an executable application from them automatically. This approach is called model-based programming and it has found application in many areas [1] [23]. The key concept here is that the models are ***domain-specific***, i.e. built from concepts related to a particular field. In the case of IPCS, this field is continuous process engineering.

Modeling paradigms and principles

The models that the process engineer creates using IPCS are capable of representing (1) the engineer's knowledge about the plant, and (2) what the necessary monitoring, control, simulation, etc. activities should do.

Every model built belongs to one modeling paradigm. IPCS supports the following modeling paradigms:

1. Process models,
2. Equipment models, and
3. Activity models.

Throughout the modeling paradigms hierarchical organization is used: models represent entities on various levels of abstraction. This hierarchical organization offers a technique for managing complexity, and was found very useful in many complex situations. General graph models could have been an alternative, but they grew to unmanageable size very quickly (especially, when visual model editing was used). For this reason, we prefer the use of hierarchical model organization. However, the use of hierarchies is not mandatory.

Process models are used for representing functionalities in the plant. They are hierarchically organized diagrams, each block in the hierarchy describing a material, energy or information transfer process. A process represents an identifiable sub-functionality of the plant.

Equipment models describe what the plant's hardware is and how it is constructed. These models are also hierarchical diagrams, each node representing a piece of hardware, be it a simple component (like a valve) or a complex assembly (like a distillation column). An equipment model represents a part of the plant's hardware.

Activity models describe computations which implement the desired monitoring, control, simulation, diagnostics and recovery functions. They are hierarchically organized ***dataflow diagrams***, each node representing a (simple or complex) activity performing some kind of data processing.

Models are created, edited, and presented in the form of various diagrams, and stored in an object database management system (ODBMS). The diagrams contain icons, connections, and various attributes associated with them. The user interacts with the models using a visual model builder tool, as described below. Models tend to be complex (simply because their domain: the large-scale plants are complex), and for this reason there are built-in capabilities in the system to manage this complexity. These capabilities are provided in the form of various organizational principles, including:

- *Hierarchy*: Models represent entities on various levels of abstraction.
- *Multiple aspects:* Models may contain many objects (icons, connections, etc.), but only those are shown simultaneously which are relevant in a given *aspect*. Aspects partition a complex model into manageable pieces. The modeling paradigm defines what aspects are available, what objects they contain, and how they interact with each other.
- *References:* Models can contain "pointers", i.e. references to other models. For example, activity models (i.e. computations) must often be tightly coupled to plant models (e.g. alarm limits may be dependent on equipment size), and this can be expressed by including a reference to an equipment parameter in the activity model for the alarm detection operation.
- *Conditionals:* Model components may be conditionalized based on the "activeness" of other components. For example, depending on what state a process is in (e.g. shutdown, startup, or running), different mathematical models describing the process must be used.
- *Types:* Models (if they satisfy certain requirements) may be converted to model types, which can be considered library components. Types can be re-used in many model configurations, and they are stored in a single, shared copy.

The application of these model organization principles makes possible the creation of very complex models for large-scale plants. Next, we look at the various modeling paradigms in detail

Plant models

In Process Models (PM), the plant's functionalities are represented in the context of the following aspects:

1. hierarchical process flow sheets (containing processes, streams, connections among them, and process variables and parameters),
2. mathematical models (containing mathematical variables and associated equations),
3. failure-propagation graphs (containing process failure-modes and their connections),
4. finite-state models (containing discrete states and state transitions), and
5. process/equipment associations (containing references to equipment models and connections between process failure-modes and equipment fault-states).

The last one is an example for references in models: a process may contain references to equipment models, and these references describe the hardware needed to realize the functionality defined by the process.

In Equipment Models (EM), the plant's equipment is described using the following aspects:

1. hierarchically organized equipment structure (containing equipment and their interconnections, and variables and parameters related to equipments),
2. finite-state models (containing discrete states and state transitions), and
3. fault models (containing anticipated equipment fault-states).

Note the distinction between functional (i.e. process) failure-modes and equipment fault-states: *failure-modes* represent how a function may fail (i.e. its requirements are violated), while *fault-states* represent the actual problem with the equipment which caused the failure. An example might be a `Storage` process with a `StorageLevelLow` failure-mode, which may be associated with the equipment Tank's `Leaking` fault-state. PM-s and EM-s together are called *Plant models*, because they let the engineers express their knowledge of the plant.

Activity models

Among the Activity Models (AM) the following model categories are available:

1. algorithmic (for user-defined subroutines and libraries),
2. timer (for generating timed pulses or delays),
3. finite-state machine (for discrete-event controls),
4. simulation (for performing real-time simulation),
5. external interface (for communicating with external datasources),
6. operator interface (for communication with the plant operator), and
7. compound (for organizing activity hierarchies).

The first ones are primitives (in the sense that they do not contain other activities), while the last one embeds other activities. Primitive activities are dataflow blocks, which include a computation to be scheduled when input data is available. The computation can be very simple (like a user-written subroutine), or very complex (like a sophisticated equation solver for a simulation block). The scheduling policy is implemented by the run-time support system of IPCS which is based on the Multigraph Computational Model (MCM) [23]. The MCM is realized through a macro-dataflow, platform-independent kernel (MCK) that allows the dynamic configuration of complex computational schemes. This feature is available in such a manner that IPCS applications can run on distributed, heterogeneous computer systems. The compound activity models are hierarchical dataflow diagrams organized from primitive and other compound activity models.

The power and practical applicability of IPCS is greatly enhanced by the use of references to process and equipment models inside the activity models. This feature makes the activities truly model-based: their behavior is dependent on the contents of the actual plant models. References in activity models to plant model components include, for example:

- references to process equations in simulation activities: they indicate in the simulation which equations should be solved,
- references to failure-modes and their connection to alarm variables in a compound activity: they indicate the sensory manifestation of the process failure-mode,
- references to process parameters and their connection to activity parameters in a compound activity: they indicate the software parameter should take the value specified by the plant model.

Note that there can be an automatic correspondence between plant models and activity models: if the plant model changes, activity models (the software which uses them) will automatically change; and an error will be flagged if the user attempts to run an inconsistent application.

One important feature of IPCS is its capability to interface with external components. There are three major interfaces available:

- *External interface* for data connection with the plant's instrumentation and (low-level) control system,
- *Operator interface* for communicating with a human operator, and
- *External data handler interface* for communicating with external packages (like sophisticated simulator systems, databanks for non-ideal chemical properties of materials, etc.). (These are an optional component and based on domain-specific installations.)

Note that the external interface to the plant's instrumentation is bi-directional, i.e. IPCS can, for instance, provide setpoints to low-level controllers. These interfaces make possible the integration of very complex systems. A typical scenario is the following: plant data is monitored and sent to a plant simulator, while some of the inputs to the simulator are specified by the operator, and the simulation results are then displayed. This way the operator can experiment with "what-if" scenarios and observe the effect of intended changes on the plant. The external components are accessed through standard, TCP/IP-based networks, so it is possible to involve systems on a global scale.

IPCS can be used for creating highly reusable models. For instance a process model of a distillation column can be developed and refined, and stored as a *type* which can be re-used at many places in a plant hierarchy. Model types can be stored in a separate database, thus forming a library of types, which can be distributed among plants of a corporation. An engineer can either use the model type as it is, or can make his/her own modified version of it.

FIGURE HERE

Figure 1: Typical model editor screen

FIGURE HERE

Figure 2: Typical operator interface screen

Implementation

IPCS is implemented in the framework of the Multigraph Architecture, which is a generic environment for building model-based programming environments [1]. In IPCS (see Figure 3), a ***graphical model editor*** provides a visual programming environment for building models. The use of graphics is not exclusive: whenever reasonable, text is used (e.g. equations). Models are stored in an ODBMS [17] which supports sophisticated datastructures and network access. A ***system integrator tool*** facilitates the construction (i.e. linking) of executable application programs. The executables contain:

1. model interpreters (to build the run-time objects from models); the run-time objects are the MCK objects which do the actual work,
2. special run-time support modules (e.g. equation solver, diagnostics reasoning engine),
3. user-defined subroutines (compiled and linked),
4. the Multigraph Kernel (for implementing the low-level scheduling paradigm),
5. the external data interfaces (for the plant data interface and other external packages), and
6. a graphical operator interface.

When an executable is started, the model interpreters create and configure the computations implementing the required functionalities. For this, they need the models stored in the database. Once interpretation is finished, the model database can be taken off-line because all the required information is present in the running code.

A typical graphical model editor screen and an operator interface screen can be seen on Figures 1 and 2, respectively. On the model editor screen, the model of compound activity `DISTILL-CMPD-ACT` is shown. Below the main diagram, two windows are available (`Browser` and `Reference`) for showing other, e.g. process, models and establishing references to the objects present there. Left to the middle, a textual attribute editor window can be seen which is used for setting the attributes of a local parameter. On the operator interface screen there are multiple windows available for (1) displaying the status of the processes (upper left corner), (2) showing the graphs of some variables and inputing values (on the left), (3) displaying the status of the equipments of the hierarchy (upper right corner), and (4) showing the currently active alarm list (lower right corner). Note that the hierarchy is defined by the models and the screen with the graphs and buttons are configured through activity models.

Practical experiences

IPCS has been installed at a polyester intermediates plant in the US, and at various other sites in Japan. There has been many, non-trivial applications built using the system, including systems for critical variable monitoring, dynamic simulation of a critical section of the plant, sensor fault detection and others, with models medium complexity (about a 2-300 different objects in the models). The model-based approach has been proven useful, for example: the application developers

FIGURE HERE

Figure 3: IPCS Implementation

could directly integrate their models of the plant with real-time data, and the operators had direct access to high-quality ("ideal") simulation models to compare the plant's behavior with. This latter capability was enormously helpful in diagnosing problems related to the dynamic behavior of processes. In another application, catalyst concentrations were predicted by a simulation using instantaneous flow and temperature measurements. (Catalyst concentrations can be measured, however, it takes about 30 mins to obtain a measured value.)

FAULT DIAGNOSTICS

We have developed a generic fault modeling and diagnosis method, and the reader is referred to [19] for a more detailed discussion. A summary is presented here.

Fault modeling commences by decomposing the plant's functionalities to generate a process hierarchy, and the plant's structure to generate an equipment hierarchy. A process model in the process hierarchy may have a set of *failure-modes* that represent anticipated deviations from its normal behavior. *Fault propagation graphs* which represent causal relationships amongst process failure-modes can be specified. A fault propagation graph contains failure-modes as vertices and the edges indicate possible propagations. A fault propagation is weighted by an estimated propagation probability and a propagation time interval (indicating the minimum and maximum expected time for the propagation to happen). An equipment model in the equipment hierarchy can have *fault-states* which describe how the equipment may fail. Propagations can also be specified from equipment fault-states to process failure-modes, thus relating the nodes of the two hierarchies.

Alarms, which are generated from sensor values using suitable algorithmic activities, can be associated with process failure-modes. Alarms are binary variables, which can be either ON or OFF, and they carry a timestamp identifying when their value changed. When an alarm becomes ON for the first time, it triggers the diagnostic reasoning engine. The reasoning system uses information related to (1) currently active (i.e. ON) alarms, and (2) structural, probabilistic, and temporal

FIGURE HERE

Figure 4: Graph of an example fault model

constraints specified by the fault-propagation graphs to generate a diagnosis result. The reasoning algorithm traces back along the edges of the fault propagation graph from the currently active alarm set to calculate the primary failure-modes of specific processes which may have caused the current alarm scenario. ***Primary failure modes*** are the ones that may have caused the current alarm sequence. The reasoning procedure is very efficient computationally: all algorithms are of polynomial complexity. An example fault model is shown in figure 4. On the graph, square blocks represent equipment fault states and circles represent process failure-modes. Dotted circles represent failure-modes which are monitored, i.e. have associated alarms. Each edge in the graph is weighted with fault propagation probability and time interval. Note that this graph appears in a different manner on the visual model builder.

Some extensions to the fault modeling and diagnosis method are as follows:

- Forewarning of critical process failure-modes. This is implemented by operating the fault model in a "forward" manner (as opposed to "backward" manner used in diagnosis): downstream propagations are calculated and impending failure-modes are signaled.

- Fault sources can be either process failure-modes or equipment fault-states. The latter case is true if there are process/equipment associations present and the fault-states are connected to failure-modes. Once these connections are built, the diagnosis results are presented in terms of actual equipment fault-states.

- The diagnostic engine can generate a specific event for each diagnosed fault source. These events can be placed into activity models, and often serve as triggers to fault recovery strategies (implemented as activities).

- Facility for explicitly modeling redundant equipments and explicitly modeling finite-state machines for activating redundant equipments. This is realized by a special class of finite-state models: in the process/equipment associations aspect of the PM-s one can introduce a finite-state automata which models the dynamically changing association of equipments to the process.

- Logs for current ON alarms and alarm event history.

These additions have been implemented in order to facilitate the specification and implementation of fault recovery strategies. A fault recovery strategy needs to know (on a timely basis) (1) the identity of the fault sources, (2) what may happen in the future, and also (3) the status of backup equipment should it decide to make a switch.

As mentioned above, the main fault diagnostics algorithms have also been optimized with regards to their execution time. It is also possible to dynamically change fault propagation probabilities and time intervals of any fault propagation link: for example, in an activity model one can utilize an algorithmic activity to calculate the parameters for a fault propagation link based on actual flow-rates and this will effect the behavior of the diagnostics reasoning mechanism. This was introduced in order to track the plant's behavior over a set of possible ranges, for example, with different production rates. Hence, fault propagation parameters can always be re-calculated whenever the need arises and all possible propagations with all possible parameters do not have to be entered into the fault model.

The basic fault modeling and diagnostics method has been used to generate diagnostics applications for power generation plants[11], chemical plants[10], and aerospace vehicles[2]. The largest real system that has been modeled this way had about 4,000 failure modes. It works well with complex and multiple sources of data by rapidly reducing volumes of information to significant results. Hence, we feel that this approach represents a generic and practical choice for addressing diagnostics problems in large-scale plants.

FAULT RECOVERY

Fault recovery is an extremely complex problem because it requires effective solutions in the areas of planning, reconfiguration, and re-scheduling. We choose not to design a *general* fault recovery method, instead, we designed the *programming environment* (IPCS) to be open-ended: it is able to incorporate additional modeling concepts and provides enough mechanisms to generate complex fault recovery applications.

When fault recovery is expected, some *response* needs to be generated by the supervisory control system. Some of the different kinds of responses that can be developed include:

- Initiating actions based on an algorithmic activity, i.e. a special-purpose algorithm. This may involve any calculations, actions introducing changes in controller setpoints, etc.
- Switching to backup equipment.
- Reconfiguring control and monitoring strategies.

The above alternatives can be realized by employing appropriate activity models, where the diagnosis results appear as events in the dataflow of the system (in the context of a compound activity model), affecting other activities.

FIGURE HERE

Figure 5: Finite-State Machine Model for switching to backup equipment

The finite-state modeling concept is one basic building block for switching to backup equipment and reconfiguring control and monitoring strategies. Users can describe different states and transitions between them and can specify the actions to be taken in case a state transition occurs. As an example, Figure 5 shows a compound activity which contains a finite-state machine (SWITCH, its details shown in the lower right corner) which models a switch operation to backup equipment. Suppose, in a plant, we have two pumps A (main), and B (backup) which support some function. In the compound activity, events are defined (A_FAULTY, B_FAULTY) which are generated by the diagnostics system whenever the corresponding pump is detected faulty. Other events (A_OK, B_OK) are derived from these, and they indicate that the respective pumps are healthy. These events trigger transitions on a finite-state machine between various operation modes (NORMAL, BACKUP, INOPERATIVE). The machine tracks the state of the system, and generates output events (MAIN_ON, BACKUP_ON) which may switch the equipment. Similarly, control and monitoring strategies can be switched to different ones upon changes in the state of a process.

Currently, the user can take advantage of the various model building facilities provided by IPCS in order to build generic activities for switching to backup equipment and reconfiguring control and monitoring strategies. Note that these strategies are implemented as high-level, visual models. The diagnostics engine provides enough information in terms of diagnosed fault sources and forewarned failures (including earliest times of failure) to be able to validate various generated actions from the viewpoints of (a) there is sufficient time for the chosen recovery strategy, and (b) there are sufficient non–faulty equipments to guarantee the execution of the chosen recovery strategy.

In detail, the diagnostics system can activate specific event variables in activity models when a piece of equipment was diagnosed as faulty or a process failure-mode was identified as the primary fault source. This event activation may trigger other activities, initiating a complex fault recovery sequence. Some possible scenarios include (but are not limited to):

- Adjustment of alarm detection limits.

- Reconfiguring the monitoring activity models (activate and deactivate parts of the activity chain).
- Initiation of a corrective control action. If no response is detected within a time-limit, a total shutdown sequence can be started.

Note that all these activities can be "programmed" using the visual model builder of IPCS.

While the above and other scenarios can easily be built using the current IPCS, more tools and concepts have to be provided to users so that they can build complex plant-wide recovery strategies. We are currently working on extending our method to include more formal means of describing planning activities, resource constraints, reconfiguration activities, and scheduling activities.

AN EXAMPLE: SENSOR VALIDATION

We present a real-life application developed for a large-scale polyester intermediates manufacturing plant and implemented in the framework provided by IPCS. The application is using similar techniques to those of [14], however (1) it employs a simpler internal representation: direct graphs instead of matrices, and (2) "gross errors" (sensor errors in our case) are detected by simple logical and statistical tests on the input data.

The problem is to validate flow and level sensors, i.e. *diagnosing* faulty sensors and *predicting* the actual readings of diagnosed faulty sensors. Multiple interlocking material balances serve as the basis for fault modeling: a material balance equation specifies a mathematical constraint among the values of sensor readings. A balance equation describes the physical fact, that outflows from a system must be equal to the sum of the inflows and the changes in levels (to accommodate accumulation, etc.). An example of material balances on a section of a hypothetical plant is shown in Figure 6. This section consists of a distillation column with its overhead feeding a tank. Three material balance equations are derived from this section (based upon the law of conservation of mass, and assuming that there are no leaks in the section):

1. Balance B1 is $F1 = F2 + F3$.
2. Balance B2 is $F1 = F2 + \Delta L1 + F4$.
3. Balance B3 is $F3 = \Delta L1 + F4$.

The flow sensors $F1$, $F2$, $F3$, $F4$, and the level sensor $L1$ are modeled such that each has a `Faulty` fault-state. Violation of balances $B1$, $B2$, and $B3$ are modeled as process failure-modes (`BV1`, `BV2`, and `BV3`). Fault propagations are modeled from the `Faulty` fault-state of each sensor to each of the balance violation failure-modes, if that sensor participates in that balance. For example, there exists a fault propagation from the `Faulty` fault-state of flow sensor $F1$ to the balance violation failure-modes `BV1` and `BV2`. Each fault propagation is weighted with 1 probability, and has the range $[0, \mathit{SamplingRate}]$ as propagation time interval. Instances of the same type of algorithmic activity are used to monitor each balance and flag an alarm if a balance is violated. These alarms are associated with their respective balance violation failure-modes. Once a sensor is diagnosed to

FIGURE HERE

Figure 6: A plant section with flows and levels

be faulty, a prediction of its true reading is made in terms of a range; from all its associated balance violation residuals and readings of other non-faulty sensors.

The real-life application has 10 material balances and 35+ sensors. Figure 7 shows the compound activity model for the sensor validation application. The activity `DMTCMBDATAVEC` is responsible for the data interface to the plant instrumentation; the block `MASSBALVEC` calculates the mass balances from the sensor readings and outputs the the residuals for each balances and a vector of binary flags indicating whether balances have been violated or not. The block `CMBDIAG` determines the status of sensors and the predicted values for faulty sensors. Each sensor is modeled as having the fault-states `ReadHi`, `ReadLo`, `StuckHi`, `StuckLo`, and `Stuck`. The fault states are detected by doing simple calculations on the input data and generating alarms whenever a discrepancy is detected. Alarms are related to failure-modes and eventually to the fault-states of the sensors. The results are communicated to the operator, and will also be communicated to the plant's DCS. On a related operator interface screen the following information is shown: (1) sensor status (faulty vs. healthy, etc.), (2) measured value (if the sensor is healthy), and (3) a range for the predicted value (if the sensor is found to be faulty, but because of redundancy, the system is able to determine upper and lower limits for the estimated values).

During the first week of its installation the application has diagnosed at least 3 faulty sensors. It seems that these and similar applications have the potential of substantial savings in plant operations. We have implemented the application as a set of generic algorithmic activities that can be instantiated for a set of material balances specified in the models. Building an application from a set of balance specifications involves (1) configuring the models, (2) wiring up the input data to the models, and (3) implementing the activities to communicate the results and potentially take corrective actions. This example demonstrates that in IPCS we can easily embed various, well-known techniques and packages, and seemlessly integrate them with plant models, custom programs, and other activity models.

FIGURE HERE

Figure 7: Activity model for the sensor validation application

COMPARISONS

It is beneficial to contrast IPCS with other approaches to the problems mentioned in the introduction. IPCS can be compared with other systems in two aspects: (1) as an environment for developing applications, and (2) as it incorporates fault-modeling and activity modeling to facilitate fault recovery.

As a *development environment*, the closest systems are real-time expert systems and application generators.

Real-time expert systems are sophisticated knowledge-based systems which operate in a real-time environment and are capable of reasoning about the process, and perform monitoring, control, and diagnostics functions. The most notable example in this category is G2 by Gensym Corp[7], which is used in various process control situations. Real-time expert systems are typically using frame-based representation forms for encoding static, relational knowledge, and rule-based representation forms to encoding actions (and dynamics). A general purpose inference engine is responsible for firing the rules, and implements a reasoning procedure.

Note that in these systems, all the knowledge must be encoded in the forms of frames and rules, which do not always belong to the everyday modeling philosophy of the process engineers. The rule interpretation process has some problems regarding efficiency (because of the inherent search and pattern matching involved in the process). In case of IPCS, the underlying execution environment,the MGK kernel, is driven by data availability and has no need for search and pattern matching.

Application generators are CASE environments with a domain-oriented flavor: they are for generating software for a particular domain. The most successful application generators are signal processing and control system design (e.g. Labview(TM) by National Instruments). The application generators synthesize executable code from high-level (mostly graphical) specifications. The domain of the application generators is restricted: control algorithm design, for example. They typically don't provide modeling facilities for modeling the plant to be controlled and their model concept set is rather limited. They do not require much software expertise, but because of their lack of rich modeling concept set, their use is limited.

To compare IPCS as a *fault diagnostics and recovery system*, we may look into at least three different system categories[15]. (1) The rule-based (or expert system-based) approaches use symptom/fault associations[8]. This approach is efficient for small plants, but for large-scale systems it runs into difficulties because of exponential algorithmic complexity, lack of temporal reasoning, and exponential growth of database size[13]. (2) Another approach based on model-based reasoning, use the plant structure and its normal functional specification to identify fault sources [4] [6] [12] [3] [21]. These methods pose problems for real-time diagnosis because of NP-hard complexity, lack of temporal reasoning, lack of facilities for managing model complexity, and the difficulty of accurately specifying plant-wide functional relationships. (3) Graph-based approaches employ causal fault models and probabilistic reasoning[20]. Various forms of these models have been used in the process industries, including fault trees[9], signed digraphs[22], event digraphs[5], Petri nets[18], and unsigned digraphs weighted with fault propagation probabilities and time intervals[16]. IPCS is closest to these approaches and is heavily utilizing the following features of this approach: (a) modeling of fault propagations, (b) the possibility of temporal reasoning, and (c) the possibility of diagnosing most commonly occurring single and non-interacting multiple fault cases in a limited amount of time.

CONCLUSIONS

The approach described in this paper is practical for generating diagnosis and recovery applications in large-scale plants because:

- It is a *generic* method for addressing diagnostics problems. Applications generated using this approach have been used for diagnosis in various domains, as mentioned above. The applications have provided accurate and timely diagnosis information from a large and complex data set.
- Multiple means of specifying *recovery strategies* are available as part of the method.
- The method encourages and supports multiple modeling paradigms including object-oriented model development. Models are hierarchical in nature, and can have multiple views, and can be reused in other applications. These features are relevant in coping with model complexity.
- This method is available as part of IPCS. Benefits in that system include object-oriented technology, database functionalities, graphical model building, and automatic application generation.

ACKNOWLEDGMENTS

The research described in this paper has been made possible, in part, by the Osaka Gas Company, and OGIS-RI, Japan. Their support is gratefully acknowledged.

Bibliography

[1] B. A. Abbott et al. "Model-based approach for software synthesis", *IEEE Software*, pages 42–52, May 1993.

[2] Carnes,J., Davis,W., Biegl, C., Karsai,G. (1991) "Integrated Modeling for Planning, Simulation and Diagnosis", *IEEE Conference on AI Simulation and Planning in High Autonomy Systems*, April, 1991.

[3] R. Davis (1983) "Reasoning from First Principles in Electronic Trouble-shooting," *Int. J. Man-Machine Studies*, vol. 19, pp. 403-423, Nov 1983.

[4] Davis,R. (1985) "Diagnostic Reasoning Based on Structure and Behavior", *Qualitative Reasoning about Physical Systems*, Bobrow.D. ed., MIT Press, Cambridge, MA., 1985.

[5] Finch, F. et al. (1990) "A Robust Event Oriented Methodology for Diagnosis of Dynamic Process Systems", *Computers in Chemical Engineering*, Vol 14, No. 12, pp. 1379-1396, 1990.

[6] Forbus,K. (1985) "Qualitative Process Theory", in *Qualitative Reasoning About Physical Systems*, Bobrow,D.ed, MIT Press, 1985.

[7] *G2 Reference Manual*, Gensym Corporation, 1990.

[8] Hayes–Roth, F. et.al (1983) *Building Expert Systems*, Addison-Wesley Publishing Company, 1983.

[9] Himmelblau, D. (1978) *Fault Detection and Diagnosis in Chemical and Petrochemical Processes*, Elsevier Scientific, Amsterdam, 1978.

[10] Karsai, G. Sztipanovits, J. et al. (1991) "A Model-based Approach to Plant-wide Monitoring, Control and Diagnostics". *Proc. of the AIChE Annual Meeting*, (microfilm) Los Angeles, CA, Nov. 1991.

[11] Karsai, G. et al. (1992) "Model based intelligent process control for cogenerator plants", *Journal of Parallel and Distributed Computing*, pp. 90–103, June 1992.

[12] Kuipers,B. (1985) "Commonsense Reasoning about Causality: Deriving Behavior from Structure", in *Qualitative Reasoning About Physical Systems*, Bobrow,D.ed, MIT Press, 1985.

[13] Laffey, T.J et al.(1988) "Real-time knowledge-based systems", *AI Magazine*, 9(1):27–45, Spring 1988.

[14] Mah,R.S., et al.(1976) "Reconciliation and Rectification of Process Flow and Inventory Data", *Ind. Eng. Chem., Process Dev.*, Vol. 15, No.1, 1976.

[15] Milne,R. (1987) "Strategies for Diagnosis" and other related papers in the same issue, *IEEE Transactions on System, Man and Cybernetics*, Vol. 17, No.3, 1987.

[16] Narayanan, N. et al. (1987) "A Methodology for Knowledge Acquisition and Reasoning in Failure Analysis of Systems", *IEEE Transactions on Systems, Man and, Cybernetics*, March/April, pp 274-288, 1987.

[17] *Objectivity Documentation, Version 2.0*, Objectivity, Inc., September 1992.

[18] Portinale,L.,Anglano,C. (1994) "B-W Analysis: A Backward reachability Analysis for Diagnostic Problem Solving Suitable to Parallel Implementation, *Application and Theory of Petri Nets 1994, LNCS 815*, pages 39-58, Springer, 1994.

[19] Padalkar,S.J. et al. (1991) "Real-time fault diagnostics", *IEEE Expert*, pages 75–85, June 1991.

[20] Reggia, J. et al (1985) "A Formal Model of Diagnostic Problem Solving", *Information Sciences*, 37, pp 227-285, 1985.

[21] E. Scarl et al. (1987) "Diagnosis and Sensor Validation through Knowledge of Structure and Function," *IEEE Trans. Syst., Man and Cybernetics*, vol SMC-17, no. 3, May/June 1987, pp. 360-368.

[22] Umeda, T. et al. (1980) "A Graphical Approach to Cause and Effect Analysis of Chemical Processing Systems", *Chem. Eng. Sci.*, vol 35, pp 2379-2388, 1980.

[23] Wilkes,M.D. et al. (1993) "The Multigraph and structural adaptivity", *IEEE Transactions on Signal Processing*, pp 2695–2717, August 1993.