MODEL BASED INTELLIGENT PROCESS CONTROL FOR COGENERATOR PLANTS

Gabor Karsai, Janos Sztipanovits, Sanjir Padalkar, Csaba Biegl
Vanderbilt University, Nashville, TN 37235, USA
Nobuji Miyasaka, Koji Okuda
Osaka Gas Company, Osaka, Japan

Abstract

This paper describes a new approach for the design and implementation of an intelligent monitoring and diagnostic system for a complex, practical system: a co-generator plant. The methodology is based on *multiple aspect modeling* and *model interpretation*, which is used for instantiating a run–time system based on a graph-model of computation. Sensory input signals from the plant are processed and interpreted in the context of various models of the co-generator system in *real-time*. The system demonstrated that the model based approach has a significant impact not only on the functional performance of the control system, but dramatically reduces development time as well.

Keywords:
distributed processing, model-based systems, real-time systems, monitoring and diagnostics, intelligent control.

Corresponding author:
Gabor Karsai
Department of Electrical Engineering
P.O. Box 1824-B
Vanderbilt University
Nashville, TN 37235
Tel.: (615) 322-2338
E-mail: gabork@vuse.vanderbilt.edu

INTRODUCTION

Increasing autonomy in industrial process plants is a central goal of automation research. The problem is specified as maintaining the ability to reliably monitor and operate plants under unforeseen circumstances such as faults or environmental changes. Conventional monitoring/control systems are usually designed by making a number of assumptions about the plant operation. But what happens if these design-time assumptions do not hold?

New research directions on "intelligent systems" attempt to address these problems. This paper discusses specific properties of intelligent automation systems based on our experience with the design and implementation of an Intelligent Process Control System (IPCS).

By *process monitoring and diagnostics* we mean a real-time process that detects anomalies in the plant and identifies their cause within a predictable amount of time. The ultimate purpose of performing this task in real-time is to increase system autonomy: the output of the monitoring and diagnostic system can be used to make preventive or corrective actions so as to maintain the plant operation in accordance with defined goals.

Intelligence is not a well defined attribute for monitoring and diagnostics systems. However one can characterize it in terms of capabilities, such as:

- evaluation and interpretation of sensory input in the context determined by the actual state and configuration of the plant,
- ability to monitor and diagnose in the presence of sensor failures,
- ability to match internal operation to actual goals by selecting among available methods and reconfiguring internal components.

The IPCS has been designed and implemented with keeping these observations in focus. However, it has not been created as a "generic intelligent system shell" without specific application: it is the result of a 4 year research effort on the development of a model based monitoring and diagnostic system for co-generator plants. Our primary purpose is to show that contrary to prevalent opinions, model based systems can:

- meet real-time requirements,
- enhance robustness and performance,
- offer a software technology that dramatically improves implementation efficiency and system testability.

The paper is organized as follows: First, we discuss four background areas: (1) the research directions in intelligent monitoring and diagnostics, (2) the target application: a co-generator plant, (3) the Multigraph Architecture that provided a software engineering environment for this project, and (4) the various design considerations that have influenced the design of the system. Next, the IPCS architecture is described, followed by the discussion on modeling and model building techniques. This is followed by a review of some example models, the run-time system, and detailed description of two specific components. The paper concludes with an evaluation of IPCS.

BACKGROUND

Research Directions in Intelligent Monitoring and Diagnostics

There are several research directions that focus on some or all of the capabilities of intelligent monitoring and diagnostic systems described above. *Expert system-based* approaches try to capture experiential knowledge about the dynamics of the plant and try to mechanically imitate the operator's reactions to certain events in the plant [8]. *Model based* systems follow a different approach: they incorporate such information into the system which is available in design time in the form of mathematical or physical models. These models are used for the validation and interpretation of sensory input signals [10], tracking changes in the state of the plant, and even for the control of the reconfiguration of the processing system [2]. Availability of models is the obvious precondition for applying the model based approach. This condition usually does not create any problem in the case of engineering systems, where the plant to be monitored/controlled has a known internal structure and operation.

The Application: Co-Generator Plant

The IPCS target application is a co-generator plant, which employs a gas turbine as prime mover, a 1MWatt electric generator, and a boiler system. The turbine burns natural gas as fuel and drives the generator that produces electricity. The waste heat energy of the exhaust gases is used for generating steam in the boiler system. The co-generator is a very efficient way of producing energy due to its utilization of the waste heat. Because the system produces two kinds of energy (heat and electricity), it is sometimes called as Total Energy System (TES).

Economical operation and complexity of the co-generator plant requires a very high level of system autonomy, specifically: (1) the plant should operate with minimal operator interaction, (2) the operator should be informed continuously about the state of the plant, and (3) if failures happen, their possible causes should be shown to the operator.

The general requirements for the IPCS of the co-generator plant called for the following features:

- continuous *monitoring* of plant operation by measuring 30 analog (continuous) and 150 digital (status and alarm) signals,
- providing real-time *diagnostics*, that involves the continuous display and revision of diagnostic hypotheses as the state of the plant changes,
- high-bandwidth *operator communication* through a graphical operator interface that provides structured access to dynamic system parameters and diagnostic results.

Multigraph Architecture

The IPCS has been implemented in the framework of the *Multigraph Architecture* (MCA). MCA was previously developed for the design and implementation of a broad category of model based intelligent systems operating in real-time environments [13],[1].

Figure 1 illustrates the multi-layer architecture of the MCA. The various layers are as follows:

- *Models and Model Builders*. The top-level of MCA-based systems includes the models, which are heterogeneous structures representing selected qualitative and quantitative aspects of both the environment (system to be controlled or monitored) and the control/monitoring system itself. The models are created using a graphical model builder tool that can be tailored for application specific modeling concepts. This and other tools constitute the *Multigraph Programming Environment* (MPE), and it can be used for creating high-level graphical programming environments that are specific to a given application domain. One such environment is the IPCS model builder environment.
- *Model Interpreters*. The multiple-aspect models embody the information necessary to generate specific *instances* of the executable components of the control/monitoring system. The process which maps the models into executable computing structures is called *model interpretation*. MCA has a general toolset and methodology to define and implement model interpreters for different application domains.
- *Execution Environment*. In order to make the model interpretation process computationally feasible (note that model interpreters transform the models into executable program structure dynamically), a special computational model is used in the Multigraph Execution Environment (MEE).

The MEE is based on a graph model of computation, where the computational graphs consist of two kind of objects: *actornodes* and *datanodes*. The actornodes are operators containing a subroutine (called the *script*) and a context table, the former being the piece of code to run when

Figure 1: The Multigraph Architecture

the actornode is scheduled, and the latter being a local memory for storing state information. The datanodes are data buffers that connect actornodes to each other. The actornodes are scheduled according to the dataflow principle: whenever data is available on their input datanodes they can execute their associated script. This algorithm can use and update the local state information (in the context) and produce data which is sent to the output datanodes. The actornodes and datanodes together form a network of processing objects which constitutes the executable program of the application.

The graph building is *dynamic* in the sense that (1) scripts for actornodes can be loaded dynamically from compiled modules, and (2) the graphs can be created dynamically by a control program. These features open possibilities for creating very flexible and dynamic systems, that, eventually, are able to change their own structure during their lifetime.

The model interpreters map the model structures into the structure of the executable programs defined by a control graph. For example, the model of a monitoring (signal processing) system stored as a block diagram is mapped into a computational graph made of actornodes and datanodes which implement the computations for the blocks. A special scheduler module, the Multigraph Kernel (MCK) provides run-time support for dynamic scheduling the elementary computations at execution time.

Figure 2: A Heterogeneous Multigraph Network

The MEE is available on a wide variety of platforms, including (1) workstations connected via a local area network, (2) tightly-coupled, shared-memory multiprocessors (SCE of IBM [5]), and (3) message-passing based multiprocessor (Transputer[1] network). The programming model provides a unified approach to parallel programming: all the architectural details are covered by the MCA programming interface. Processing networks can be built in a heterogeneous manner: parts of the graph can run on different processors. Because the assignment of actor nodes to processors is done by the model interpreter/graph-builder program, the elementary algorithms can be developed independently of the parallel hardware used. Heterogeneity is supported: for example the graph builder control program and a part of the processing graph can run on a workstation which communicates with a transputer-based multiprocessor using the facilities provided by the MCK, and other parts of the graph are allocated on a set of transputer nodes. (Illustrated in Figure 2.)

MCA offers a very natural environment for a broad range of applications and it has been successfully applied in building various intelligent systems, e.g. experiment builder for an NMR instrument [11], intelligent test integration system [12], and reconfigurable signal processing systems [15].

[1] *Transputer* is a trademark of INMOS, Ltd.

DESIGN CONSIDERATIONS

The design and implementation of IPCS has been influenced by the following considerations.

- *Incorporation of models.* In accordance with the general methodology of the MCA, the model-based approach should be followed: models are used to represent the plant and the control system.
- *Model based programming environment.* Construction of complex models requires a programming environment which offers domain specific concepts, abstractions and composition principles, and provides interactive tools for incremental model-building.
- *Automatic system integration.* Because the models contain information about the components of the monitoring/diagnostic system itself, the same models should be used for configuring those systems. The system integration heavily relies on the dynamic linking/loading facilities of MCA.
- *Real-time performance.* The run-time system should avoid using time-consuming search and memory management techniques and should provide predictable response times.

IPCS ARCHITECTURE

The IPCS architecture is divided into two main components, the *Development Shell* and the *Target Shell*.

Development Shell

It was recognized early in the project that model based systems require special attention for building and maintaining the models, since most of the complexity is concentrated within them. The IPCS Development Shell is a model building environment for editing the models of the monitoring and diagnostic system. Figure 3 illustrates the structure of the Development Shell. The System Editor component consists of various specialized model builder and editing tools for creating and incrementally specifying the various models of the system. *Incremental model* building is a critical aspect of the technology: it conforms to the actual work-style of design engineers who build specific monitoring/diagnostic system applications. The process engineers can directly interact with the system. Therefore, there is much less need for a separate software development activity then in the case of traditional systems.

The output of the Development Shell is the result of the model building process: a set of models which represent the plant and its associated monitoring/diagnostic system.

Figure 3: The Structure of the Development Shell

Target Shell

Analyzing the nature of different monitoring/diagnostic system applications, one can recognize that they have approximately the same general structure in execution time. The differences between specific systems are fully determined by the information included in the models. The Target Shell is an abstraction that defines the generic structure (or *skeleton*) of monitoring and diagnostic systems as is illustrated in Figure 4. Components of the skeleton system are:

- *Physical Interface:* couples the IPCS to an external Data Acquisition System, receives and distributes the incoming analog and digital data among the components of the monitoring and diagnostic system.

- *Monitoring System:* includes processing modules (scaling, computing values of process variables, fault detection algorithms, etc.), and real-time database points for storing measured or computed process variables.

- *Diagnostic System:* interprets the incoming and computed alarms and provides fault diagnostics in real-time.

Figure 4: The Structure of the Target Shell

- *Operator Interface:* provides structured access to the real-time database points and to the current set of diagnostic hypotheses.
- *Alarm Simulator System:* generates a real-time alarm sequence corresponding to a selected fault in the plant. The alarm simulator is used as a testing tool for the diagnostics.

Each IPCS implementation incorporates these main components, and in each implementation they are differently configured on the basis of the information contained in the application specific models.

MODELING AND MODEL BUILDING

The first step in the design of model based systems is to answer the following questions:

- What are the modeling concepts to be used?
- What are their relationships to each other?
- What is the most appropriate representation system?

Model-based monitoring and diagnostic systems must include extensive information about the plant and their own internal structure. Monitoring requires knowledge about physical processes (material and energy flows) because they represent the basic *functionalities* in the plant and determine the abstraction levels for how the plant operations are to be presented to the operator. Monitoring also requires modeling of *information flows* (e.g. dataflows in processing systems that compute process variables and check various constraints to detect failures), and models for operator communication. The diagnostics system uses fault models for representing (1) fault modes and (2) fault propagation paths, but also utilizes the representation of the (3) physical component structure of the plant, because the result of fault diagnostics has to be expressed in terms of physical components.

Models

The models that are created during the model building process are complex structures representing different aspects of the environment, the control system and the interactions between the two. Note that by employing models instead of hand-coded algorithms the complexity of the system is dominantly structural, and the code complexity is typically negligible. This implies that standard, efficient, and well-tested algorithms can be utilized everywhere. For each new application, only its models have to be developed and the the algorithms can remain the same. This feature significantly shortens the time and required for system development.

The structural complexity problems of the models (i.e. that they have simply too many items and relationships to keep track of) can successfully be attacked by using a technique called *multiple-aspect* model building. The models describe the plant from different *aspects*, use different abstractions (e.g. about the physical structure and/or the plant dynamics), and are typically organized into one or more decomposition hierarchies controlling the level of details shown. Obviously, these modeling aspects can not be independent from each other; the ultimate purpose of extensive modeling is exactly the representation of these interactions. Therefore, the modeling methodology has to define the relationships between these aspects together with consistency criteria.

Characterization of objects from different aspects is a well-known method in modeling. There are AI tools (e.g. ART, [3]) that directly support the creation of "multiple views" on a reasoning process (e.g. tracking hypothetical situations). According to our experience, the real difficulty is not the representation of different aspects, but the expression of the interactions among them. The critical question is how to facilitate the well-structured representation of these interactions? MPE tackles this problem by allowing the declaration of *structurally independent* (SI) and *structurally dependent* (SD) modeling aspects for the model building tools.

Two modeling aspects are SI if the model structures do not impose direct constraints on each

Dominant Aspect	**Dependent Aspect**
Process Model Structural Aspect	State Transition Aspect
	Fault Propagation Aspect
	Monitoring and Control Aspect
	Operator Interface Aspect
	Simulation Aspect
Physical Model	

Table I: Modeling Principles of IPCS

other. If the modeling aspects are SD, the structure of the so called *dominant* aspect determines the structure of the *dependent* aspects. In other words, the dominant aspect provides a set of constraints and properties that the dependent aspects inherit.

An important issue of modeling is the representation form. The two most often proposed solutions are: (1) rule form possibly augmented with numeric representation schemes, and (2) declarative representation languages defined according to the nature and concepts of the selected modeling disciplines. According to our and other [4] experiences, rule-oriented representation methods are often inefficient, intractable, and in a number of cases are not sufficient for general modeling problems. The declarative representation languages offer a much better solution but, their use is quite awkward if the models are large. The selected solution in MPE is *graphic model representation with hierarchical decomposition* which serves as a very high level interface to declarative representation languages. This graphical methodology was applied to the IPCS model building approach as well.

Further, declarative languages offer an excellent opportunity for automatic test and validation [14]. The basic approach for doing testing and validation in a declarative framework is as follows: (1) the declarative language forms are mapped into a unified graph structure, (2) test and validation criteria are defined for the different modeling aspects, (3) the criteria are expressed as graph properties, and (4) graph algorithms are used to check the properties. This methodology supports the automatic test of the consistency of the individual modeling aspects and the consistency among the SD aspects. A serious limitation of the test approach is that only static properties of the models can be tested this way. Using a new research direction, we will be addressing the problem of testing the dynamic run-time behavior of the system.

Modeling Concepts

The basic modeling principles used in IPCS and their relationships are summarized in Table I.

The *Process Model* is a dominant aspect that describes the energy and material transfer processes

in the plant. The process model can be considered as a functional representation of plant operation since it defines those concepts that can be used to specify the required operating conditions.

The *Structural Aspect* is the main aspect of the Process Model that contains structural information about the process. The *State Transition Aspect*, *Fault Propagation Aspect*, *Monitoring and Control Aspect*, *Operator Interface Aspect*, and *Simulation Aspect* are dependent aspects to the structural aspect. The reasons are: (1) the state transition aspect represents the possible state transitions and triggering events in the process, (2) the fault propagation aspect represents anomalies in terms of the required functionalities (also specified in the structural aspect), (3) the monitoring/control aspect specifies (a) the information flows necessary to calculate those process variables that can not be observed directly but are also defined in the process models, and (b) the associated control loops, (4) the operator communication must be based on the abstractions defined in the process model, and (5) the (optional) simulation aspect represents algorithms which can generate simulated data for testing the monitoring and control system. Indeed, the structure of the process model imposes structure and provides context for all four of the dependent aspects.

Thus, a process model, PM, describes a functional entity of the system from many aspects at the same time. Note that the representation includes not only the plant to be controlled, but also the process control system responsible for that control. A PM is the union of the various aspects of the model:

$$PM = \bigcup(PM_{Struct}, PM_{State}, PM_{Mon}, PM_{Fault}, PM_{Oint}, PM_{Sim})$$

The two most important aspects, the structural an fault propagation aspects are briefly reviewed below.

PM_{Struct}, the *Structural Aspect* of the process model, describes the material, energy and information transfer taking place in the process in the form of 5–tuples:

$$PM_{Struct} = (Iv, Ov, S, SP, Lv, Cn)$$

where Iv, Ov, S, SP, Lv, denote the input and output process variables, the states of the process, the subprocesses, and the local (internal) process variables, respectively. Cn describes the set of connections between process variables and processes. The process is considered as a functional entity that takes the values of the input process variables and determines the values of the output process variables based on the values of local process variables. The process may be modeled in terms of its subprocesses and how they connect to the process variables (input, output or local) of the parent process. This decomposition can be recursively applied, hence the processes in a system form a hierarchical representation. Note that the process variables and subprocesses are *abstract entities* and do not play a role during the run–time of the system. The structural aspect

of the process model is the Structurally Independent aspect. The other aspects are all Structurally Dependent aspects.

PM_{Fault}, the *Fault Propagation Aspect* of the process model, represents the process from the aspect of failures and failure propagation in a process model hierarchy. For this purpose, it introduces the concept of a *failure mode*. Failure modes represent certain faulty states of the process and their relationship to the failure(s) of physical components of the process. This aspect consists of 7–tuples

$$PM_{Diag} = (FM, AO, CL, FA, SP, PhFM, FProp)$$

where the various sets are as follows:

- FM denotes the set of failure modes associated with a process.
- AO denotes the set of alarm objects related to the process. Activation of an alarm object indicates that the process is in a failure mode. These alarm objects are also modeled in PM_{Mon}.
- CL denotes the set of *causal links* which represent the failure propagation. Each causal link has three parameters: (1) the minimum propagation time, (2) the maximum propagation time, and (3) the propagation probability.
- FA denotes the set of failure mode–alarm associations. If a failure mode is associated with an alarm, the alarm being active represents that the process is in the associated failure mode.
- $PhFM$ denotes the set of physical component–failure mode associations. It simply gives a list of names of physical components the failure of which can introduce the associated failure mode.
- SP denotes the set of subprocesses for the parent process. This is the same as in the case of PM_{Struct}.
- $FProp$ denotes the set of connections which describe the possible failure propagations. Failures always propagate along causal links. They can go from one subprocess to another subprocess, or from a subprocess to one of the failure modes of the process. The former kind of connection represents how failures propagate from one subprocess to another subprocess, the latter one describe how failures in subprocesses influence the behavior of the parent process.

The *Physical Model* is a dominant aspect that represents the spatial decomposition of the plant. The physical component hierarchy specifies assemblies, sub-assemblies, elementary components, independently from their actual functions in the plant operation. The same components may serve

multiple functions or, there may be components without any functional assignment at a specific time instance. Due to this, the physical model is structurally independent from the process model. The structural independence does not mean that relationships do not exists between the two hierarchies. Logical links between processes and physical components can be represented and are used in the the diagnostic reasoning.

Graphical Model Building Environment

Modeling requires tools for representing the models. The representation technique has to satisfy two contradictory requirements. First, the representation system must provide an "interface" for the model designer, i.e. the represented model has to be easily comprehensible by humans. Second, the represented model has to be machine readable since the models constitute the "knowledge-base" which determines the system operation. Based on these requirements and on the fact that models express dominantly structural information, MPE supports two equivalent representation forms: declarative languages and the corresponding graphic representation. Models may be expressed in symbolic terms however, graphical ways are preferred due to the apparent expressiveness of visual representation [6].

The model building environment is fully graphical and supports the incremental building of models in terms of SD and SI aspects. For example, after starting the model builder to edit the fault propagation aspect of a process model, an initial version of the fault propagation graph is automatically extracted from the process model based on the already defined structural aspect. This initial model will then be further refined by the model designer.

The designer builds the models using icons: placing them on a drawing board, connecting them, etc. The builder tool then synthesizes the symbolic structures required in the Target Shell, performing various checks and validations in the meantime. The designer is free to look at (or even edit) the symbolic models, but this activity is not encouraged.

The Development Shell facilitates model building in terms of *graphic structures* that constitute models of the plant on different levels of abstractions. Whenever an abstraction level has been identified, a corresponding icon should be created which will represent it on one level higher. The model changes are automatically kept track of and the designer is informed whenever an inconsistent change has been introduced. Any change in the models has to be introduced by modifying the graphic structure.

The physical component hierarchy is represented in the form of a set of Physical Model declarations that are similar to the Process Model declarations. The relationships between the physical model hierarchy and the process model hierarchy are expressed in the form of links between the nodes of the two trees.

Figure 5: The Structural Aspect of the Circulation Process Model

The IPCS/TES Models

Figure 5 and Figure 6 illustrate two aspects of a process model. Figure 5 illustrates the *Structural* aspect of the *Circulation* process, wherein its internal structure and interactions are defined. This process consists of four subprocesses, *Afterburner*, *Temperature Control*, *Pump*, and *Heat exchanger*. Additionally, it has the following set of input process variables: *WaterIn*, *GasIn*, *WasteIn*, and *ControlIn*. These are set by the "external" world. The output process variables (*WasteOut*, and *WaterOut*, are set by the process for the external world. Process interactions are modeled by connecting process variables to processes. Note that this process also has internal variables (e.g. *Water*) that are used for modeling internal process interaction (e.g. between the *Pump* process and the *Heat exchanger* process).

Figure 6 illustrates the failure propagation diagram for the same process. Note that the same set of subprocesses are present, since this is just another aspect of the same process. However, they are represented by different icons. The picture represents how local failures propagate from one process to another one, e.g. the *Exhaust Temp-hi* failuremode of the *Temperature Control* process will introduce an *Exhaust Gas Temp. High* failuremode in the *Heat exchanger* process. The link is established via a causal link (in this case *cl17*) contains propagation parameters.

The Process Model together with its dependent aspects is represented in the form of a process

Figure 6: The Fault Propagation Aspect of the Circulation Process Model

model declaration. This declaration is *automatically* generated from the drawing. Note that the graphical model from a particular aspect might be considered as a special way of looking at one part of this declaration. Figure 7 illustrates parts of the *Circulation* process declaration.

Table II illustrates the complexity of the IPCS models for the co-generator plant. The development of the model took 4 man-months. In our judgement, this is considerably faster than conventional methods for developing control systems of this complexity.

Currently, the models are represented in a Lisp-based language and the model interpreters are written in Lisp. Figure 8 illustrates the process model hierarchy as it is shown to the operator.

Monitored Analog Signals:	301
Monitored Digital Signals:	1501
Number of Processes:	62
Number of failure propagation graphs:	15
Average number of failure modes per process:	6
Size of (symbolic) model files:	app. 1.6MBytes
Size of executable system:	app. 5MBytes

Table II: Model Sizes for the Co-generator Plant

```
(DEF-PROCESS |Circulation|
 ;; Input/output process variables
 (WaterIn WasteIn GasIn AirIn ControlIn
  -> WaterOut WasteOut)
 ;; Process failure modes
 (|Steam Flow Low| |Steam Flow High| |Steam Water Flow Low|
  ... |Exhaust Gas Pres. Hi|)
 ;; Internal process variables
 (PVARS Water Waste)
 ;; States, transitions, etc.
 (STATES (|Operational| NIL) (|Valves Open| NIL))
 (STATE-DEPS (|Valves Open| ... ))
 (TRANSITIONS) (PARAMETERS)
 ;; Signals, events and alarms associated with the process
 (SIGNALS (|Analog Signal| SIGNAL)) (EVENTS)
 (ALARMS |Circ. Exhaust Gas Temp. Hi| |No. 3 Evp. Diff. Pres. Lo|)
 ;; Data acquisition interface
 (INTERFACE ((CEGTH-IFP ALARMIFP) -> |Circ. Exhaust Gas Temp. Hi|)
 ...
 ;; Fault propagation graph
 (FAULT-GRAPH
   (((|Temperature Control| |Exhaust Temp-hi|))
     (|cl17| (|Valves Open|) 10 6 15)
    (|Heat exchanger| |Exhaust Gas Temp. High|)))
 ;; Operator panel for this process
 (PANEL (|Circulation Panel| (|Circulation-proc| .... )))
 ;; Subprocesses of the Circulation process
 (SUBPROC
  (|Afterburner|
   (|Afterburner| (|WasteIn| |GasIn| |AirIn| |ControlIn| -> |Waste|)))
  (|Temperature Control|
   (|Temperature Control| (|Waste| -> |WasteOut|)))
  (|Pump| (|Circulation Pump| (|WaterIn| -> |Water|)))
  (|Heat exchanger|
   (|Heat Exchanger-II| (|Water| -> |WaterOut|)))))
```

Figure 7: An example process model declaration: Circulation Process

Figure 8: The Process Hierarchy of IPCS/TES

TARGET SYSTEM

As we have mentioned before, a target system is an instantiation of the Target Shell. The instantiation is performed by a process called *multiple-aspect model interpretation*. Model interpreters belonging to each component of the skeleton system traverse the model structures and build the internal structure of the executable system by creating the control graph for the execution environment in the meantime. The significance of this technique is the following:

- *Automatic system integration*. Components of the target system are generated from the same, consistent model set *automatically*. The automatic integration nearly eliminates the time and effort needed to integrate a specific application. Having a run-time library for the Target Shell components, development of a new system is practically reduced to the graphically supported model building.

- *Modification*. Similarly, modification of the monitoring and diagnostic system as a response to changes in the plant configuration requires changes only in the models. Effects of these changes will be automatically propagated to the target system in the model interpretation process.

- *Speed.* In the MCA, the control-flow of the model interpretation and the real-time processing systems of the target system are separated; connection (and the necessary synchronization) is created only when the target system has to be fully or partially reconfigured. As a result of this solution, the run-time speed of the target system components is comparable to that of the conventional systems. In the current implementation, the target system is linked from C program modules.

Run-time System Architecture

The Target Shell is implemented as two major components:

- Model Interpreter and Execution Environment,
- Operator Interface.

The first component contains the various model interpreters that process the symbolic models and create MCA computational graphs in the Execution Environment. These graphs are interfaced to a data acquisition system providing communication with the plant instrumentation hardware. According to the facilities provided by MCK, these networks can exist in one Unix process, in many processes on the same (multitasked) CPU, or in many processes residing on different CPUs connected through some reliable communication network (e.g. Ethernet). All the scheduling of the processing nodes is done under the control of the MCK scheduler.

The execution environment consists of two main components: the monitoring system and the diagnostics system. The diagnostics system uses efficient graph-algorithms for reasoning [9] on the fault propagation graphs contained in the process models. Figure 9 illustrates a simplified diagram of the monitoring and diagnostics run-time system.

The Operator Interface component is responsible for all operator interface communication. It is built according to the operator interface models, and it provides various facilities for user input and output. It is interfaced to the execution environment using standard interprocess communication mechanisms. All the user interaction abstractions were implemented using standard X window system widgets. We found that because of the event–driven nature of the X window based programs it was very appropriate to implement the operator interface as a separate process rather than incorporating it into the execution environment.

The monitoring and diagnostics system created by the model interpreters is highly parallel. Because all the computations are expressed in the form of the Multigraph network, the computations on parallel branches of the graph can be executed concurrently. In fact, we have made experiments by parallelizing the diagnostics algorithms and ran them on a multiprocessor (IBM 3CE), while the rest of the system was executing on a workstation. (Illustrated in Figure 10.)

Figure 9: The Run-Time System Architecture

Figure 10: Parallel Diagnostics and Monitoring

EXPERIENCE

The model based system building approach has proved to be extremely flexible and efficient. All of the models of the co-generator application have been built in approximately 4 man-months by persons who were not software engineers.

The system runs on an HP 9000 Series 300 workstation which is connected to an HP data acquisition system. The model interpreters are implemented in Lisp. All run–time modules are implemented in C. Graphical support is provided by the X window system.

The system underwent field test in May 1989, in Osaka, Japan, and has been in continuous use since that time. The real-time performance is satisfactory. The system is able to handle the real-time monitoring and diagnostic functions with the model complexity described before. In a simulation wherein all the alarms in the co-generator plant's models were turned on at the same time, the system required 10 seconds to begin diagnosis at the root of the model and complete it at the bottom.

There were two important observations worth mentioning:

1. The graphical model building should be supported by sophisticated tools, otherwise graphical editing might be a very laborous activity.

2. Graphical interaction tools are needed for the operator for interacting and controlling the process. The operator should be able to pose "WHAT-IF"-type of questions to the system, and foresee the possible effects of operator actions.

A newer version of the system has been built which exhibits these new features.

EVALUATION

It is interesting to note how IPCS compares against to the requirements for a real-time AI system as specified in [7].

- **Efficient integration of numeric–symbolic processing.** It is naturally solved in IPCS: the MCA seamlessly integrates the two environments.

- **Continuous operation.** IPCS monitors 30 continuous signals, without any problems with garbage collection, because the run-time modules are written C.

- **Focus-of-attention mechanism.** The diagnostics system interprets the alarms in the context of the hierarchical process model and it does a refinement on the diagnostics results as it proceeds down the tree.

- **Interrupt-handling facility.** All the alarm signals which activate the diagnostics algorithms are asynchronous. When a new alarm arrives the diagnostics procedure is restarted.
- **Optimal environment utilization.** MCA provides facilities for this: the computational graphs may have priorities assigned to them which can be changed dynamically.
- **Predictability.** The diagnostics algorithms do not allocate memory dynamically, hence there is no need for garbage collection. The algorithms are also of polynomial complexity: response times are bounded and predictable.
- **Temporal reasoning facility.** The diagnostics algorithm takes the fault propagation delay information present in the fault propagation graphs, thus it employs temporal reasoning.
- **Truth maintenance facility.** The diagnostics procedure is triggered whenever a new set of alarms arrives and it starts a diagnostic reasoning for that part of the model which is related to the new alarm. This means that assertions are dynamically created and retracted based on the actual alarm set information and process state information.

CURRENT STATUS AND FUTURE PLANS

Currently, this experimental prototype is being beta-tested in many process plants, in the US, as well as in Japan and Europe. The plants are from the chemical and energy industries. Initial results indicate the feasibility of the model-based approach in these environments.

Future plans with the system include:

- *Re-implementation in C++.* This is required for efficiency and maintainability reasons.
- *Improvements in the graphical model builder.* A very crucial component, the graphical model builder, essentially determines the usability of the system. There are several improvements planned.
- *Increase in efficiency.* Currently, mostly because of the Lisp implementation, the models tend to be large and the model interpretation process can take a long time. With a more efficient, C++-based implementation we expect this situation be improved.
- *Object-oriented database for model storage.* As the complexity of models increases so does the need for a mechanism for efficient model representation and storage. An object-oriented database seems as a suitable solution for this problem, and we plan to incorporate such a component in the system.

- *Simulation facility*. In many practical situations, it is required to have an on-line simulation model of the plant, where the operators can perform "experiments" on the dynamic model of the plant, before actually introducing changes in the real system. The models will be extended to incorporate systems of nonlinear algebraic equations and differential equations which can be used for synthesizing a simulation system through a code-generation process. An experimental prototype has been implemented and successfully tested at one of the beta-test sites.

At the time of this writing a system with the above features is being designed and implemented.

CONCLUSIONS

This paper described a model based approach in the design and implementation of an intelligent real-time monitoring and diagnostic system for co-generators. The proposed approach is based on the application of the MCA architecture which combines modeling, model interpretation and execution into a unified framework. Experiences obtained during the development and the field test of the IPCS system have proved that the approach is viable even in real-time applications, and results in dramatic improvement in software technology as well.

Bibliography

[1] Biegl, C.: "Design and Implementation of an Execution Environment for Knowledge-Based Systems", Ph.D. Thesis in Electrical Engineering, Vanderbilt University, 1988.

[2] Blokland, W., Sztipanovits, J.: "Knowledge-based Approach to Reconfigurable Control Systems", *Proc. of the American Control Conference*, pp. 1623-1628, 1988.

[3] Clayton, B.C.: *ART Programming Tutorial*, Inference Corp. , 1985.

[4] Davis, R.: "Form and Content in Model Based Reasoning", in *1989 Workshop on Model Based Reasoning at IJCAI-1989*, pp. 11-27. 1989

[5] Garcia, A. and Freitas, R.F. "The 8CE Multiprocessor Workstation", *IBM Symposium on Parallel Processing*, Poughkeepsie, October, 1988.

[6] Karsai, G.: "Declarative Programming Techniques for Engineering Problems", Ph.D. Thesis in Electrical Engineering, Vanderbilt University, 1988.

[7] Laffey,T.J. et al. "Real–Time Knowledge–Based Systems", *AI Magazine*, pp. 27-61, Spring, 1988.

[8] Moore, R.L. et. al.: "Expert System Applications in the Industry," *Proc. of the ISA International Conference*, pp. 22-25, Houston, TX, 1984.

[9] Padalkar,S, Karsai, G, Biegl, C, Sztipanovits, J: "Real-time Fault Diagnostics Using Hierarchical Fault Propagation Models", *IEEE Expert*, pp. 75-85. June 1991.

[10] Scarl, E.A. et. al.: "Diagnosis and Sensor Validation through Knowledge of Structure and Function", *IEEE Trans. on Systems, Man, and Cybernetics*, pp. 360-368, May-June, 1987.

[11] Sztipanovits, J., Biegl, C., Karsai, G., Bourne, J., Mushlin, R., Harrison, C.:"Knowledge-Based Experiment Builder for Magnetic Resonance Imaging (MRI) Systems," *Proc. of the 3rd IEEE Conference on Artificial Intelligence Applications*, Orlando, FL, pp. 126-133, 1987.

[12] Sztipanovits, J., Karsai, C. et al.: "Intelligent Test Integration System," *Proc. of the Conference on Artificial Intelligence for Space Applications*, Huntsville, AL, pp. 177-185, 1986.

[13] Sztipanovits, J., Biegl, C., Karsai, C., "Graph Model-Based Approach to Representation, Interpretation and Execution of Real-Time Signal Processing Systems", *International Journal of Intelligent Systems*, pp. 269-281, Vol 3., No.3 1988.

[14] Sztipanovits, J., Padalkar, S., Krishnamurthy, C., Purves, R.B.: "Testing and Validation in Artificial Intelligence Programming", *Proc of the 3rd SPIE's Conference on Space Station Automation*, Cambridge, MA. 1987, pp 2-10.

[15] Wilkes, M., Lynd, L., Sztipanovits, J. and Karsai, C., "The Multigraph Approach to Parallel, Distributed, Structurally Adaptive Signal Processing," *Proceedings of the 1990 International Conference on Acoustics, Speech, and Signal Processing*, Albuquerque, April, 1990, pp. 2037-2040.