Error Specification, Tracking and Recovery in Computer Integrated Manufacturing

A. I. Kokkinaki

Computer Science Department,
University of Cyprus,
Nicosia, Cyprus.
csaik@turing.cs.ucy.ac.cy

K. P. Valavanis
A-CIM Center, and
The Center for Advanced Computer Studies,
University of Southwestern Louisiana,
Lafayette, LA 70504.
kimon@cacs.usl.edu

Abstract

State transitions in Computer Integrated Manufacturing Systems result either from the execution of the planned actions, or from the occurrence of unpredictable events. One key requirement for task planning designed for CIM systems, is the ability to monitor, or track events, so as to respond to them appropriately. Related issues include the specification and classification of possible system faults, errors and failures which may be provoked by various event occurrences. Certain constraints pertinent on event tracking in CIM systems are analyzed in this paper and a classification of faults, errors and failures is included. Furthermore, an error specification language is designed, a recovery mechanism is proposed and some working examples are presented.

1 Introduction

Planning methods designed for static environments, assume fully predictable effects of the executed actions, unlimited resources and perfect information about the domain. These assumptions do not capture certain characteristics [15] of Computer Integrated Manufacturing (CIM) systems. In CIM systems, state transitions may result either from the execution of the planned actions, or from the occurrence of unplanned events.

In CIM systems, an event is similar to the concept of a process in qualitative process theory [10]. That is, an event is something that acts over time to change some parameters of the system. However, in CIM systems, there is the additional requirement that an event does not belong to the predefined set of system actions, which transform the system from one stage to some other, upon their successful execution. In automated manufacturing environments, unplanned events can not be completely eliminated, regardless of the enforced safety features. In many cases, elimination, or prevention of all these events may not be critical, or feasible. As noted in [19], events may be instigated by other agents in the system, or just occur (i.e. lighting bolt).

Some events may not affect the overall system execution, whereas some others may instigate faults, errors and failures in the system affecting the plan execution, altering some actions effects in the system etc. To act in a realistic manner, a task planning system must, among other things, monitor the events in its environment, so as to respond to those events that affect the plan execution. The ability to monitor the events in the system is also referred as event tracking. The concept of events (primitive and composite) has also been examined in other research areas like Active and Temporal Databases [11, 17].

In a CIM system, faults are events which have been sensed in the system. A fault may be a physical defect, environmental stress, operator's mistake, etc. Without loss of generality, in CIM systems, some faults are expected; that is, it is anticipated that something may go wrong. Moreover, faults are unpredictable, in the sense that there is no deterministic specification on what type the next fault will be, in which subsystem it will occur, or when it will take place. However, some faults may be modeled a-priori. That is, the system designers may include in the system model "provisions" for certain classes of faults. Features of the modeled faults include parametrized information about the influence the faults exert on the system. This paper focuses on modeled faults only, and the term fault is used to mean a "modeled fault" (a-priori) from now on. Fault detection and identification are not issues included in the scope of this paper;

that is, it is assumed that there exists the underlining structure which is necessary to detect and identify faults. In general, faults may be distinguished into: i)*Permanent faults*, which remain in existence until a specific recovery action is taken, ii) *Transient faults*, which appear and disappear, usually without causing irreversible system damage, and, iii) *Intermittent faults*, which appear and disappear repeatedly.

An error is the manifestation of a fault, that is, a detected discrepancy between the actual and the expected state of the system. The error is identified with some delay (from the time the fault occurred until the time the error was identified). This delay may be minimal and it can be estimated based on the detection capabilities of the system. In view of the above classification of faults, errors may also be distinguished into permanent, transient and intermittent. It is assumed that only permanent errors require the initiation of a recovery procedure. That is, transient errors do not need to be recovered and intermittent errors may not be recoverable.

The term "error recovery" is frequently overloaded to signify different concepts. Error recovery in automation may follow the emerging standard terminology from the area of "reliable computing" [5], as pointed out in [6]. Error recovery and failure handling was primarily considered from the perspective of situation assessment, followed by locally triggered reactive operations [1, 2, 9, 13]. As outlined in [4], it is important for a CIM system, to provide a framework for on-line dynamic planning system which must be capable of replanning in case an unexpected situation arises, and to plan opportunistically, that is, to have the flexibility to reach the same goal state in different ways, depending on the availability of system resources.

Every identified error has some characteristics which may be parametrized. Error parameters include, but are not limited to: i) the time the error was identified, ii) the agent(s) affected by the identified error, and iii) the action (or actions, if any) which was affected by it.

The *extent* of an error refers to the agents in the system that have been affected by that error. Depending on the extent of the error, errors may be distinguished into: i) *Isolated errors* (only one agent affected), and ii) *Segregated errors* (more than one agents affected).

The *span* of an error refers to the actions which have been affected by that specific error. Depending on the span of the error, errors may be distinguished into: i) *Null point error* (no action affected), ii) *Single point errors* (only one action affected), and iii) *Multiple point errors* (more than one actions affected).

It is interesting to note that an isolated error may not be a single point error. To elucidate this, consider the following case. Assume that an agent has been affected by an error while it was executing an action. Moreover, assume that the affected action has synergetic dependencies on some other action(s) executed in parallel by some other agent(s). Therefore, although only one agent has been affected by the error, more than one actions have been affected by the error's occurrence. Similarly, a segregated error might be a null point error, that is, an error affects some agents, all of which were idle at that particular time instant.

Finally, depending on its span, an error may be proclaimed to be a failure. That is, a *failure* is an error with a span that includes one or more actions in the current plan. Failures are critical; the initiation of their recovery procedure must have the highest priority. The recovery procedure for a failure, or an error may result into a normal state (if the recovery is successful), or in the fatal state, as shown in Figure 1.

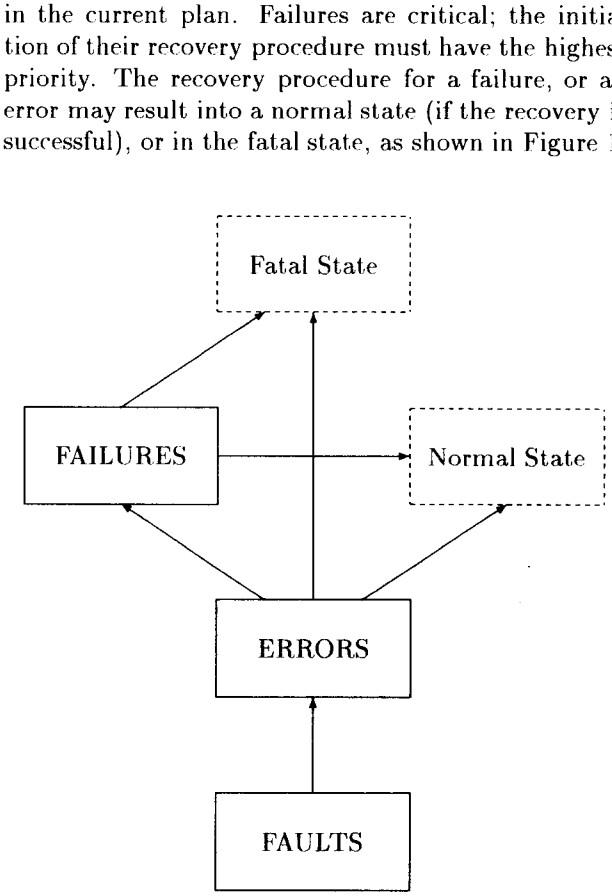

Figure 1: Classifications of abnormalities

The rest of this paper is organized as follows: Section 2 presents a formalization of the above discussion. The proposed error specification language and

an error recovery mechanism are presented in Section 3. Section 4 presents some working examples while section 5 concludes this paper.

2 Definitions of Faults, Errors and Failures

In this section, we present the definitions of faults, errors and failures in the system; an extented analysis may be found in [16].

Definition 1: The set $S = \{s_1, s_2, \cdots, s_{|S|}\}$ denotes the set of agents in the system. Every agent is capable of performing (sequentially) a specific set of actions over a period of time. It is assumed that there is complete knowledge about the agents capabilities and their individual sets of actions.

Definition 2: The set $A = \{a_1, a_2, \cdots, a_{|A|}\}$ denotes the set of actions performed by the system agents.

Definition 3: The set $F = \{f_1, f_2, \cdots, f_{|F|}\}$ denotes the set of the modeled faults in the system. A *fault* is an unplanned/unanticipated change in the system. Every fault occurrence is associated with the time instant at which the fault came into existence.

Definition 4: The set $E = \{e/e$: system error $\}$ denotes the set of all possible errors in the system. An *error* is a discrepancy between the actual and the expected state of the system.

Every error is associated with certain characteristics which may be parametrized. An analysis of these parametrized characteristics follows.

Definition 5: The *extent*, Υ_e, of an error $e, e \in E$, in the system, is defined in terms of the set of agents which are affected by the occurrence of the error e. That is,

$$\Upsilon_e = \{x/x \in S, x \text{ affected by } e\}$$

- An error $e, e \in E$ is called isolated, iff $|\Upsilon_e| = 1$.
- An error $e, e \in E$ is called segregated, iff $|\Upsilon_e| > 1$.

Definition 6: The *span*, Ψ_e, of an error $e, e \in E$, in the system, is defined in terms of the set of actions which are affected by the occurrence of the error e. That is,

$$\Psi_e = \{x/x \in A, x \text{ affected by } e\}$$

- An error $e, e \in E$ is called a null point error, iff $|\Psi_e| = 0$.
- An error $e, e \in E$ is called a single point error, iff $|\Psi_e| = 1$.
- An error $e, e \in E$ is called a multiple point error, iff $|\Psi_e| > 1$.

Definition 7: The *timestamp* of an error $e, e \in E$ is denoted as t_e and it is defined as the point of time the error e was detected.

In CIM systems, the following constructors may be used to specify composite errors. Similar concepts have also been applied in Active Database systems to support event detection [11, 12]. Using these constructors, errors may be defined in terms of primitive errors and/or composite errors.

Definition 8: The composite error $e_i \vee e_j$, is called *disjunction of errors* e_i and e_j, where $e_i, e_j \in E$. The disjunction of errors e_i and e_j occurs when either error e_i, or error e_j occurs. It is true that:

$$\Upsilon_{e_i \vee e_j} = \begin{cases} \Upsilon_{e_i} \text{ if only } e_i \text{ has occurred} \\ \Upsilon_{e_j} \text{ if only } e_j \text{ has occurred} \\ \Upsilon_{e_i} \bigcup \Upsilon_{e_j} \text{ if both } e_i \text{ and } e_j \text{ have occurred} \end{cases}$$

$$\Psi_{e_i \vee e_j} = \begin{cases} \Psi_{e_i} \text{ if only } e_i \text{ has occurred} \\ \Psi_{e_j} \text{ if only } e_j \text{ has occurred} \\ \Psi_{e_i} \bigcup \Psi_{e_j} \text{ if both } e_i \text{ and } e_j \text{ have occurred} \end{cases}$$

$$t_{e_i \vee e_j} = \begin{cases} t_{e_i} \text{ if only } e_i \text{ has occurred} \\ t_{e_j} \text{ if only } e_j \text{ has occurred} \\ min(t_{e_i}, t_{e_j}) \text{ if both } e_i \text{ and } e_j \text{ have occurred} \end{cases}$$

Definition 9: The composite error $e_i \wedge e_j$, is called *conjunction of errors* e_i and e_j, where $e_i, e_j \in E$. The conjunction of errors e_i and e_j occurs when both errors e_i and e_j have occurred, regardless of order. It is true that:

$$\Upsilon_{e_i \wedge e_j} = \Upsilon_{e_i} \bigcup \Upsilon_{e_j}$$

$$\Psi_{e_i \wedge e_j} = \Psi_{e_i} \bigcup \Psi_{e_j}$$

$$t_{e_i \wedge e_j} = max(t_{e_i}, t_{e_j})$$

Definition 10: The composite error $e_i; e_j$, is called *sequence of errors* e_i and e_j, where $e_i, e_j \in E$. The sequence of errors e_i and e_j occurs when first error e_i and afterwards error e_j have occurred. It holds that:

$$\Upsilon_{e_i; e_j} = \Upsilon_{e_i} \bigcup \Upsilon e_j$$

$$\Psi_{e_i;e_j} = \Psi_{e_i} \bigcup \Psi_{e_j}$$

$$t_{e_i} < t_{e_j}$$

$$t_{e_i;e_j} = t_{e_j}$$

The following three constructors are also referred as filters of the errors. These filters are set depending on how many times a specified error pattern has been identified during a specified time interval $[t_i, t_j]$. The default time interval for the filters is from the time the system starts until the time the system accomplishes its goals, that is, the interval $[t_0, t_f]$. However, it is advisable to use the default time interval only when it is absolutely necessary, because the detection of the following filters is computationally expensive.

Definition 11: The filter of an error $e, e \in E$ denoted by the $\odot$ constructor, that is, $\odot e$ IN $[t_i, t_j]$, is set only once, after the first detection of e even if multiple detections of e occur during the specified time interval $[t_i, t_j]$. It is true that:

$$\Upsilon_{\odot e \text{ IN } [t_i,t_j]} = \Upsilon_e$$

$$\Psi_{\odot e \text{ IN } [t_i,t_j]} = \Psi_e$$

$$t_{\odot e \text{ IN } [t_i,t_j]} = t_e{}^1$$

where $t_e{}^1$ is the timestamp of the first detection of the error $e, e \in E$.

Definition 12: The filter of an error $e, e \in E$ denoted by the $\otimes$ constructor, that is, $n \otimes e$ IN $[t_i, t_j]$, is set when the error e has been detected n times during the specified time interval $[t_i, t_j]$. In the degenerate case, this filter is set every time $(n = 1)$ the error e is detected (alarm situation). It holds that:

$$\Upsilon_{n \otimes e \text{ IN } [t_i,t_j]} = \Upsilon_e$$

$$\Psi_{n \otimes e \text{ IN } [t_i,t_j]} = \Psi_e$$

$$t_{n \otimes e \text{ IN } [t_i,t_j]} = t_e{}^n$$

where $t_e{}^n$ is the timestamp of the n^{th} detection of the error $e, e \in E$.

Definition 13: The filter of an error $e, e \in E$ denoted by the $\neg$ constructor, that is, $\neg e$ IN $[t_i, t_j]$, is set when the error e has not been detected in the specified time interval $[t_i, t_j]$. It is true that:

$$\Upsilon_{\neg e \text{ IN } [t_i,t_j]} = \emptyset$$

$$\Psi_{\neg e \text{ IN } [t_i,t_j]} = \emptyset$$

$$t_{\neg e \text{ IN } [t_i,t_j]} = t_j$$

Definition 14: When the following sequence of errors

$$e; \neg e \text{ IN } [t, t + \tau]$$

is identified, where: $t_e = t$, $t_{\neg e \text{ IN } [t,t+\tau]} = t + \tau$, and $\tau < \epsilon, \epsilon \in R^+, \epsilon \to 0$, then this sequence of errors is called a *transient* error.

Definition 15: When the following disjunction of errors

$$\odot(n \otimes (e; \neg e \text{ IN } [t, t + \tau])) \text{ IN } [t, t + \tau'] \vee$$

$$\odot(n \otimes (e; \neg e \text{ IN } [t, t + \tau])) \text{ IN } [t, t + \tau']; e$$

is identified, where: $t_e = t$, $t_{\neg e \text{ IN } [t,t+\tau]} = t + \tau$, and $\tau < \epsilon, \epsilon \in R^+, \epsilon \to 0, \tau' < \epsilon', \epsilon' \in R^+, \epsilon < \epsilon', \epsilon' \to 0$, and $n > 1$ then, this disjunction of errors is called an *intermittent* error.

Definition 16: A *failure* is defined as an error $e, e \in E$, (either primitive, or composite) that has a span Ψ_e which contains one, or more actions forming the current plan. That is, if the current plan includes actions $\{a_i, a_j, \cdots, a_m\}$ while

$$\Psi_e \bigcap \{a_i, a_j, \cdots, a_m\} \neq \emptyset$$

then, e is a failure.

Definition 17: We define as *delay* the amount of time elapsed from the time the fault occurred until the time the error (or failure) was detected.

3 Error Specification and Recovery

In view of the previous discussion, an error specification language describes how errors, constructors and filters of errors may be combined to describe complex erroneous situations.

Grammar Γ_1 outlines a language for the description of erroneous situations. However, Γ_1 is an ambiguous grammar. For example, grammar Γ_1 can have more than one parse trees generating the string $e_1 \wedge e_2; e_3$. The described erroneous state may result i) when the conjunction error $e_1 \wedge e_2$ is followed by e_3, or ii) when the sequence $e_2; e_3$ is conjugated with the identification of the error e_1. Therefore, it is imperative to define precedence relations among the error constructors and filters. In accordance with the Boolean Algebra, the constructors are evaluated left to right. In the following precedence hierarchy, constructors are presented in order of decreasing precedence; constructors at the same precedence level are shown at the same line.

$\neg, (), \odot, \otimes$

$\wedge$

;

$\vee$

With the enforcement of these precedences, the grammar Γ_1 is transformed into grammar Γ_2. However, grammar Γ_2 contains left recursions; that is the leftmost symbol on the right hand side of the production is the same as the symbol in the left hand side of the production and this may lead the language parser into infinite looping. The elimination of left recursions from Γ_2 results into the final grammar Γ_3.

GRAMMAR Γ_1

error $\rightarrow$ error C error | (error) | filter |fault

C $\rightarrow \wedge | \vee |;$

filter $\rightarrow$ T error IN [time, time]

T $\rightarrow \odot |$ number $\otimes | \neg$

number $\rightarrow$ number digit | digit

time $\rightarrow$ number : number : number

digit $\rightarrow 0|1|2|3|4|5|6|7|8|9$

fault $\rightarrow f_1|f_2|\cdots|f_{|F|}$

GRAMMAR Γ_2

error $\rightarrow$ error $\vee$ temp1 | temp1

temp1 $\rightarrow$ temp1 ; temp2 | temp2

temp2 $\rightarrow$ temp2 $\wedge$ temp3 | temp3

temp3 $\rightarrow$ (error) | filter | fault

filter $\rightarrow$ T error IN [time, time]

T $\rightarrow \odot |$ number $\otimes | \neg$

number $\rightarrow$ number digit | digit

time $\rightarrow$ number : number : number

digit $\rightarrow 0|1|2|3|4|5|6|7|8|9$

fault $\rightarrow f_1|f_2|\cdots|f_{|F|}$

GRAMMAR Γ_3

error $\rightarrow$ temp1 error$'$

error$' \rightarrow \vee$ temp1 $|\epsilon$

temp1 $\rightarrow$ temp2 temp1$'$

temp1$' \rightarrow ;$ temp2 $|\epsilon$

temp2 $\rightarrow$ temp3 temp2$'$

temp2$' \rightarrow \wedge$ temp3 $|\epsilon$

temp3 $\rightarrow$ (error) | filter | fault

filter $\rightarrow$ T error IN [time, time]

T $\rightarrow \odot |$ number $\otimes | \neg$

number $\rightarrow$ number digit | digit

time $\rightarrow$ number : number : number

digit $\rightarrow 0|1|2|3|4|5|6|7|8|9$

fault $\rightarrow f_1|f_2|\cdots|f_{|F|}$

3.1 An Error Recovery Mechanism

An error recovery mechanism in the system may be modeled as a deterministic automaton. Depending on the identified error and the state of the system, a recovery procedure may be initiated. The definitions follow.

Definition 18: The state of the error recovery mechanism at any instant, t, denoted by $\phi(t)$, is an element of a finite set Φ.

Definition 19: The recovery action, that is the output of the error recovery mechanism at the instant, t, denoted by $R(t)$ is an element of the finite set of the recovery operations in the system, P.

Definition 20: The identified error, that is the input of the error recovery mechanism at the instant, t, denoted by $e(t)$ is an element of the set of the system errors, E.

Definition 21: The transition function $F(\bullet, \bullet)$ determines the state of the error recovery mechanism at time $t+1$, in terms of the state and the identified error at time t. That is,

$$F : \Phi \times E \rightarrow \Phi$$

and

$$\phi(t+1) = F(\phi(t), e(t))$$

Definition 22: The output function $G(\bullet)$ determines the recovery action at any time instant t, in terms of the state at that time instant t. That is,

$$G : \Phi \rightarrow P$$

and

$$R(t) = G(e(t))$$

Definition 23: The error recovery mechanism is defined by the quintuple

$$\{\Phi, E, P, F(\bullet, \bullet), G(\bullet)\}$$

In this section, the derivation of grammar Γ_3 was provided. Γ_3 is an unambiguous grammar without left recursions, which may be used for the description of various system errors. The derived error descriptions are based on the known set of faults and a given set of error constructors. For every identified error, the recovery mechanism presented in this section may describe some recovery procedure. To illustrate these concepts, we present some working examples in the following section.

4 A Case Study

The following examples are based on the configuration of the Robotics and Automation Laboratory (RAL) within USL. RAL has three robots with attached vision systems and two conveyor belts arranged as shown in Figure 2. The Adept1 is dedicated to conveyor one and the PUMA robot is dedicated to conveyor two. The Adept3 can serve either conveyor one or two and it is the only robot capable of picking up heavy items. It is assumed that faults may occur in a non-deterministic way. However, some faults can be modeled in the system. A non exhaustive but representative list of possible faults in the system follows.

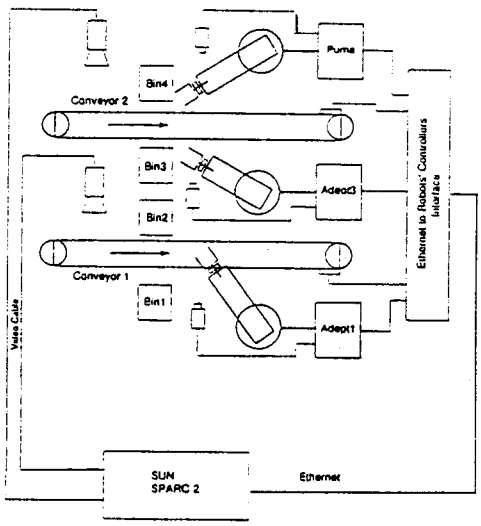

Figure 2: RAL Configuration

- f_1 : Conveyor belt one is out of order.
- f_2 : Conveyor belt two is out of order.
- f_3 : Adept1 is out of order.
- f_4 : Adept3 is out of order.
- f_5 : PUMA is out of order.
- f_6 : The global vision camera on conveyor belt one failed to identify an incoming object.
- f_7 : The global vision camera on conveyor belt two failed to identify an incoming object.

• f_8 : The local vision camera on Adept1 failed to identify an incoming object.

• f_9 : The local vision camera on Adept3 failed to identify an incoming object.

• f_10 : The local vision camera on PUMA failed to identify an incoming object.

Therefore, the set of faults in the system is $F = \{f_1, f_2, \cdots, f_{10}\}$. When fault f_i has been detected, then we say that error e_i has been identified, or alternatively error e_i has occurred. Assume the following recovery procedure for the following identified errors:

• e_1 : Adept1 stops working;
Adept3 is set to its minimum velocity.

• e_2 : PUMA stops working;
Adept3 is set to its minimum velocity.

• e_3 : Adept3 is set to its maximum velocity.

• e_4 : Adept1 is set to its maximum velocity;
PUMA is set to its maximum velocity.

• e_5 : Adept3 is set to its maximum velocity.

• e_6 : No Operation.

• e_7 : No Operation.

• e_8 : No Operation.

• e_9 : No Operation.

• e_10 : No Operation.

Assume that the sequence $e_1; e_3$ has occurred. That is, a fatal error on conveyor belt one has been followed by a fatal error on PUMA. Using the situation assessment and reactive recovery method when e_1 is identified Adept1 is set to stop working and Adept3 is set to work at a minimum speed. Similarly, when e_3 is identified, Adept3 is set to work at a maximum speed. However, this is not necessary, because Adept3 needs to handle objects in just one conveyor belt, that is conveyor belt two. In the proposed approach, however, we are able to specify that the composite error $e_1; e_3$ is a special case in the system and it should have the same recovery procedure as error e_1.

Assume that the conjunction $e_3 \wedge e5$ has occurred. That is, both Adept1 and PUMA went out of order. The reactive recovery mechanism would provide an acceptable recovery procedure, that is to operate Adept3 at a maximum speed. Even though Adept3 operates at its maximum capabilities, it is still possible that it misses some of the incoming objects. In our approach, it is possible to specify that the composite error $e_3 \wedge e5$ must have as its recovery procedure the operation of both conveyor belts at their minimum speed and the operation of Adept3 at its maximum speed. Notice that the same result would be possible with the reactive recovery approach, if the recovery procedures for errors e_3 and e_5 are modified to include the operation of the corresponding conveyor belt at a minimum speed. However, those modifications would have as a result the suboptimal operation of the system, if only a single error (either e_3, or e_5) occurs.

5 Summary and Conclusions

In Computer Integrated Manufacturing Systems, prevention or elimination of unpredictable events may not be feasible. The occurrence of these events may instigate system errors or failures. One key requirement for task planning designed for CIM systems is the ability to track events and to respond to possible erroneous states.

In this paper, a formal language for error specification was derived, a classification of errors was presented and a recovery mechanism was provided.

References

[1] Bastos, J. M., "Batching and Routing: Two Functions in the Operational Planning of Flexible Manufacturing Systems", *European Journal of Operational Research*, Vol. 33, pp. 230-244, 1988.

[2] Bertolotti, E., "Interactive Problem Solving for Production Planning", *AI Applications in Manufacturing*, Famili, A., Nau, D. S. and Kim, S. H. Eds., AAAI Press, 1992.

[3] Carberry, S., "Incorporating Default Inferences into Plan Recognition", *Proceedings of the National Conference on Artificial Intelligence*, pp. 471-478, 1990.

[4] Darbyshire, I. and Davies, B. J., "EXCAP: An Expert Generative Process Planning System", *Proceedings of the IFIP WG 5.2 Working Conference on Knowledge Engineering in Computer Aided Design*, Gero, J. S, Ed., Amsterdam: North-Holland, pp. 291-303, 1985.

[5] Decker, K. S., Garvey, A. J., Humphrey, M. A. and Lesser, V. R., "Control Heuristics for Scheduling in a Parallel Blackboard System", *International Journal Pattern Recognition Artificial Intelligence*, Vol. 7, no 2, 1993.

[6] Doyle, R. J., "A Distance Measure for Attention Focusing and Anomaly Detection in System Monitoring", *Proceedings of the Twelveth National Conference on Artificial Intelligence*, 1994.

[7] Dousson, C., Gaborit, P. and Ghallab, M., "Situation Recognition: Representation and Algorithms", *Proceedings of International Joint Conference on Artificial Intelligence*, pp. 166-172, 1993.

[8] Durfee, E. and Lesser, V. R., "Using Partial Global Plans to Coordinate Distributed Problem Solvers", *Readings in Distributed Artificial Intelligence*, Bond, A. H. and Gasser, L., Eds., Morgan Kaufmann Publishers, Palo Alto, CA, 1988.

[9] Firby, R. J., "An Investigation into Reactive Planning in Complex Domains", *Proceedings of the Sixth National Conference on Artificial Intelligence*, San Diego, CA, pp. 59-69, 1988.

[10] Firbus, K., "Qualitative Process Theory", *Artificial Intelligence*, 24, pp. 85-168, 1984.

[11] Gatziu, S., "Events in an Active Object-Oriented Database System", *Proceedings of the First Workshop on Rules in Database Systems*, Edinburg, 1993.

[12] Geppert, A., Gatziu, S. and Dittrich, K. R., "Rulebase Evolution in Active Object-Oriented Database Systems: Adapting the Past to Future Needs", *Computer Science Department, University of Zurich, TR-95-13*, 1995.

[13] Kautz, H. A., "A Circumscriptive Theory of Plan Recognition", *Intentions in Communication*, Cohen, P. R., Morgan, J. and Pollack, M. E. Eds., MIT Press, Cambridge, MA, 1990.

[14] Keller, R. M., "The Role of Explicit Control Knowledge in Learning Concepts to Improve Performance", *Machine-Learning Technical-Report-7*, Rutgers University, New Brunswick, NJ, 1987.

[15] Kokkinaki, A. I. and Valavanis, K. P., "On the Comparison of AI and DAI Based Planning Techniques for Automated Manufacturing Systems", *Artificial Intelligence in Industrial Decision Making, Control and Automation*, S. Tzafestas and H. Verbruggen Eds., Kluwer Academic Publishers, 1994, pp. 569-627.

[16] Kokkinaki, A. I., "A Dynamic Planning System for Automated Manufacturing Environments", Ph.D. Thesis, The Center for Advanced Computer Studies, USL, Lafayette, 1995.

[17] Pissinou, N., Snodgrass, R., Elmasri, R., Mumick, I., Ozsu, M., Pernici, B., Theodoulidis, B., "Towards an Infrastructure for Temporal Databases", *ACM Sigmod Record*, Vol. 23, no. 1, pp 36-61, 1994.

[18] Song B. F. and Cohen, R., "Temporal Reasoning during Plan Recognition", *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pp 247-252, 1991.

[19] Tambe, M. and Rosenbloom, P. S., "Event Tracking for an Intelligent Automated Agent", *Time94 An International Workshop on Temporal Representation and Reasoning*, 1994.

[20] Van Beek, P. and Cohen, R. "Resolving Plan Ambiguity for Cooperative Response Generation", *Proceedings of International Joint Conference on Artificial Intelligence*, pp. 938-944, 1993.