Tunghai Journal 36(3)1995:137-152
July, 1995

Knowledge representation of an execution error recovery module for shop floor control

Hsien-Jung Wu*

ABSTRACT

In manufacturing systems, error recovery is typically treated as an individual activity, ignoring the integration with other activities required for the control system. A representation scheme of error recovery knowledge is required to integrate the control software and error recovery techniques. A methodology for knowledge representation of the error recovery module for shop floor control is presented. The primary objective of the methodology is to establish the critical link between the error recovery module and control software. This link is obtained by creating a close connection with the execution module for the error recovery module which is a part of the control software. The methodology proposed is inherently generic and the representation schem covers different levels in a hierarchical manufacturing system. (This paper presents part of work discussed in [Wu & Joshi, 1995]).

Introduction

The current trend in manufacturing is to develop flexible systems capable of accommodating errors which are defined as the "inconsistency that makes the expected results of preplanned actions incomplete". One required characteristic of robust and flexible manufacturing is an error recovery strategy to recover from errors and resume preplanned actions. The development of an error recovery module has been examined in different ways: on-line or off-line, human intervention or automatic, and different degrees of combination among them. Basic guidelines for the development of the error recovery module include operator control, redundant components, program enhancement, and automatic recovery. Reflecting these guidelines for creating an error recovery module, different error recovery techniques have been developed for specific purposes.

*Department of Industrial Design Tung-Hai University, Taichung, Taiwan

Depending on the application and the complexity of system involved, different error recovery techniques can be utilized in the error recovery module. This module should provide error detection, identification, and diagnosis required to perform error recovery. In particular, in terms of control software, this module should be a part of the shop floor control system (SFCS). Unfortunately, error recovery is typically handled as an individual activity. Error recovery techniques which have been developed are hardly integrated with control software generation. The lack of integration causes additional effort in transforming recovery techniques into executable recovery procedures for use in a control software.

To integrate the error recovery module with control software, the development of the methodology entails the following problems:

- a structure for the error recovery module regarding control software,
- a knowledge representation scheme and reasoning that is suitable for building recovery knowledge,
- a process of generating recovery procedures in an incremental manner.

Review of relevant work

Error recovery techniques have been broadly developed in different manufacturing domains such as robotics, machining, assembly, and general manufacturing systems. To deal with error recovery, Abu-Hamdan et al. presented an overview of the subject of error recovery in a manufacturing system which includes uncertainty and types of errors, the sensory system and their integration with control system, and the application of AI techniques [Abu-Hamdan et al., 1990]. In addition, another point of view was that the characteristics of error recovery in a manufacturing system were different from recovery for a single component, difficult to enumerate, and more than just replanning [Moon et al., 1989].

In regard to these characteristics, research interest has been in the areas of error classification, error detection, knowledge representation, reasoning, diagnosis, recovery procedure generation, and intelligent error recovery. Failure analysis using fault tree is commonly applied in error recovery. Examples can be found in [Srinivas, 1977], [Kokawa et al., 1983], [Hassapis et al. 1987], [Chaar, 1990]. Knowledge based systems are another popular technique utilized in error recovery such as [Gini & Gini, 1983], [Lee et al., 1983], [Freyermuth, 1992], [Borchelt et al., 1994]. Other techniques and tools have been developed to deal with error recovery: augmented program [Smith et al., 1992], reconfiguration [Adlemo et al., 1992], dynamic rescheduling [Tsukada & Shin, 1993], and Petri nets [Jafari, 1990]. Knowledge representation plays an important role in the diagnostic stage of building the error recovery module. In addition to using knowledge bases, several approaches have been proposed: fault-tree analysis [Narayanan et al., 1987], scheme-base representation [Kumaradjaja et al., 1989], and linguistic modeling [Tam, 1989].

However, to design a control system with recovery ability for large discrete event systems like manufacturing systems requires not only error recovery techniques but also a complete error recovery module which can be embedded into control software. From the implementation point of view, the development of the error recovery module attempts to bridge the gap between control software and error recovery techniques. The methodology proposed in this paper includes a knowledge representation scheme using grammatical rules. This methodology does not intend to develop new recovery technique, instead it focuses on the representation required to integrate the error recovery module with control software. In particular, a knowledge acquisition and reasoning procedure is presented to deal with errors across control levels. A hierarchical manufacturing system is used as a target environment. A structure of the error recovery module is briefly discussed to support the development of this method.

Structure for the error recovery module (ERM)

The main purpose of building the error recovery module is to: 1) monitor task execution, 2) detect errors, and 3) provide diagnostic advice and generate recovery procedures. The ERM consists of an error preprocessor, a recovery procedures generator, and an interface. This paper focuses on the detail of the preprocessor.

The error preprocessor is the first component of ERM. On-line error preprocessing is the key to the error recovery module. The error preprocessor enable the ERM to detect various errors based on the attributes of the tasks along with sensory information. The Preprocessor utilizes expertise provided by the knowledge base to assist in error detection. In the knowledge base, it stores several grammatical rules converted from the task list. The error preprocessor consists of four major components: 1) the task-execution monitor, 2) the error parser, 3) the error detector, and 4) the knowledge base.

Basically, the task-execution monitor receives a list of tasks and sensory information requested by preconditions or postconditions of the tasks specified by the execution module. The values of sensory information are then passed to the error parser to compare actual sensory values with desired values. The error detector is responsible for interpreting these comparisons to detect and report possible errors based on a series of inference processes supported by the knowledge base. In case multiple errors are detected simultaneously, the priority of recovery from errors should be decided by the detector with the assistance of the knowledge base. The output of the error detector is then transferred to the next component of ERM: the recovery procedure generator.

Whenever an error is detected, the recovery procedure generator is responsible for generating recovery procedures if no suggestion of correcting action is provided by the knowledge base. Based on the error situation, the generator performs the following: 1) select appropriate built-in recovery procedure to execute, 2) generate modified or new recovery procedures, and 3) update information for the purpose of error prevention. These three activities are

performed based on the recovery techniques discussed in Section 2, attribute of execution task, and degree of recovery. The task attribute and recovery degree are described in next section.

Knowledge representation

I. Errors classification

To detect errors adequately, clear classification of error types is a critical step. However, literature review shows that there is very little description of what types of errors related to controller task execution will occur in manufacturing environment. Typical cases of errors discussed in previous works were machine failure, tool breakage, deadlock, software error (programming error), and human operation error. However, actual errors occurring in a manufacturing system are far more complicated and variant, which makes the classification of errors difficult and confusing.

The more error categories defined, the more corresponding detection methods are required. For example, several situations may occur in terms of "collision" category, such as collision between machine and part, and that between machine and machine. In this case, there should be different recovery procedures for each kind of collision. Current error classification has no apparent connection with the corresponding detection method for each error category. Details of error detection processes need to be well defined. To avoid complex detection processes, the solution is to classify errors as simply as possible. The purpose is to simplify knowledge representation and the process of generating recovery procedures corresponding to certain errors in the later stage.

Another concern is that current error classification treats error recovery as an independent activity for which no connection with the controller was provided. The classification of errors needs to reflect the attributes of controller tasks. Due to the variety of manufacturing devices, it is difficult to list all possible errors, and it may not be realistic to make manufacturing systems

capable of identifying and recovering from all errors. Also, the cause of some errors is not always obviously analyzed for diagnostic use, and the extent to which error can be detected and identified is based on several factors such as system configuration, cause-effect knowledge, and cost. Therefore, it is helpful to classify errors in a simple and generic manner that makes detection of execution errors generic and controller-task oriented.

Since the input of the ERM is a list of controller tasks specified by the execution module, the attributes of controller tasks are first addressed. The attribute of a controller task is either message passing (no equipment operation involved) or operating equipment (physical operation involved). A controller task is associated with its preconditions and/or postconditions which describe controller's situations before and after task execution respectively. Standing at the current state, the controller is expected to move to the target state if the task is completely executed. Errors may occur 1) before or 2) during the task execution. Errors occurring before task execution indicate preconditions of the task are not satisfied as requested, while those occurring during task execution show postconditions of the task are not achieved due to the task failure. To deal with execution error, therefore, two simple categories of errors are classified as precondition error (precondition unmatched) and task error (postcondition unmatched). These two error categories reflect the situations described above. This taxonomy offers a transparent classification of errors which is probably detected during execution of controller tasks. In this manner, all error types described by other researchers can be characterized as either precondition errors or task errors form the controller's standpoint. In this manner, at each level's controller, the categories and definition of these two errors are similar and listed as follows:

a. precondition error: when the evaluation value of the precondition of controller task is false.
b. task error: when the evaluation value of the postcondition of controller task is false.

II. Grammatical rules

Modeling knowledge about the failure behavior or malfunctions of a manufacturing system is a critical step of developing the ERM for shop floor control. The knowledge base in the error preprocessor provides not only knowledge required for error detection but also that required for the following recovery procedure generation. In general, the knowledge base includes the attributes of the tasks, desired sensor value, and domain expertise (e.g., error symptoms, possible causes of a symptom, etc.). Modeling knowledge of errors in a computer integrated manufacturing system is important for error detection and subsequent diagnostic activities required to recover this system from errors. To avoid further manufacturing systems slowdowns, the development of knowledge representation and reasoning about errors should provide a simple way of detecting errors, locating error source and determining appropriate recovery procedures.

Examples: An equipment controller of a robot has a controller task: Pick_up_part. Then, the required knowledge may include the following information:

1) attribute of task: operating equipment.

2) domain expertise(can be represented as a fault-tree)[Lee et al. 1983]:

Pick_up_part	-- location error	--	feeder empty or jammed
		--	part defective
	-- approach error	--	collision with part
		--	missed part
	-- grasp error	--	no part
		--	several part

The domain expertise in the example described above shows error types and possible causes for the task (Pick_up_part). However, no predefined recovery procedure is reported by this domain expertise. In this case, the knowledge

representation scheme should provide the ability of updating knowledge for use in the subsequent stage whenever recovery procedure is available. In order to build the knowledge base, knowledge representation and reasoning about errors are expected to be clearly and easily connected with the controller tasks and sensory information in which normal situations are defined. From a practical standpoint, a transparent knowledge representation and reasoning approach will easily detect errors and generate suitable recovery procedures that enables the controller to have the capability of error recovery. Usually, parameters of knowledge representation determine the degree of difficulty in reasoning which influence the process of detection and generation. The number of parameters of knowledge representation of errors should be simplified but made generic enough to represent all possible cases. As a consequence, the result of reasoning about errors will enable the system to detect error type, locate error source and determine proper diagnostic advise in an effective manner.

All of the knowledge representation schemes discussed in previous work represent error knowledge in different formats, and provide some ways of reasoning about errors to produce diagnosis actions. However, for different domains of manufacturing systems, specific parameters required to build knowledge representation schemes are not well-defined which makes the knowledge acquisition process complex. Furthermore, in a system consisting of large amounts of components, using methods such as fault tree analysis increases the problem of complexity.

Due to the structure of the ERM and the role of the knowledge base in the error preprocessor described in this paper, knowledge representation and reasoning should consider the following factors:

1) Parameters required by knowledge representation should reflect error classification which has been previously defined.

2) Recovery procedures should reflect the attributes of the controller tasks.

3) A formalization of knowledge representation and reasoning is required to cover different levels of controllers and different classes of controllers at the same level in order to simplify the process of detection and generation.

In order to establish the knowledge base, the following assumptions are made: 1) the number and the types of required sensors are known, 2) symptoms and their possible causes are predefined (failure reason analysis such as: there are 26 possible causes of tool breakage [Moon et al. 1989]), and 3) preconditions and postconditions of the controller tasks are predefined. To simplify knowledge representation and to avoid the expansion of knowledge base, a formal representation of grammatical rules is presented. These grammatical rules have the following format:

$PE \rightarrow \neg\ PRT_{ij}$ SR_k

$TE \rightarrow \neg\ POT_{ij}$ SR_k

$UE \rightarrow UMT_r$ SR_k (only for workstation and shop level)

where

PE: precondition error,

TE: task error,

UE: unsolvable or external error,

PRT_{ij}: precondition i of task j (default is TRUE),

POT_{ij}: postcondition i of task j (default is TRUE),

UMT_r: unsolvable or external error message of task r received from the next lower level,

SR:semantic routines associated with specific grammatical rule,

i=1,2,...a (i is the number of precondition),

j=1,2....b (j is the number of controller task), and

k=1,2,...c (k is the number of semantic routine).

This format of grammatical rules formalizes and simplifies the knowledge representation by concentrating on the controller tasks and their attributes. These rules can be rapidly created as long as assumptions are matched and domain expertise is provided in advance. For example, to represent grasp error if gripper is not open, the fault-tree of "pick_part" shown in previous section can be translated into grammatical rules as follows:

$E \rightarrow \neg\ PRT_{11}$ SR_1 (PE is a grasp error)

$TE \rightarrow \neg\ POT_{11}$ SR_2

where

PRT_{11}: gripper_open for pick_part,

POT_{11}: part_in_intermediate_position for pick_part,

SR_1:{report_precondition_error,
report_possible_cause_is_gripper_broken},

SR_2:{report_task_error,
suggest_recovery_procedure_modify_program}.

The first rule represents a precondition error if "gripper_open" is false and the semantic routine reports error and possible cause; the second rule indicates a task error if "part_in_processing_position" is false and the semantic routine reports error and suggests a recovery procedure of modifying program. An example of grammatical rule for external or unsolvable error is:

UE→UMT1 SR3 (in workstation level controller)

where

UMT_1: can be an unsolvable message about task 1 of the equipment level controller, and

SR3: { system_reconfiguration, operator_intervention }.

Each semantic routine may have several subroutines based on knowledge acquired at the stage of building these rules. Possible subroutines are reporting error, identifying cause, and suggesting recovery procedure. Since the grammatical rules are created through user interface for domain expertise, they are viewed as a static knowledge representation. A semantic routine may be empty if the acquired knowledge cannot afford information about a specific error. For example, there are undetected errors due to 1) sensors are not enough and 2) some errors are not significant at specific time period. In this case, additional condition checking and semantic routines will be dynamically and incrementally appended by updating the knowledge base through the interface of the ERM.

III. Generating Grammatical Rules from the task list

Procedure Grammatical-Rule-Conversion

1) Read the input from the task list.
2) For a controller task in the task list, find its attribute and label this attribute.
3) Record preconditions, postconditions, and built-in recovery procedures.
4) Create the task table.
5) Create the rules for precondition errors.
6) Create the rules for task errors.
7) Create the rules for unsolved errors.
8) Repeat step 2) to 7) for next task.

The major purpose of this procedure is to retrieve values of three parameters presented in previous section from the task list. In this procedure, a task table including parameter values of tasks is used as an intermediate representation and each task is represented in the following format:

```
# task_name task_attribute
* condition_checking
  recovery_procedure recovery_degree
```

recovery_procedure recovery_degree

.....

To use the task of "grasp" as an example, results are shown as follows:

Task: grasp
attribute: Operating equipment
precondition: gripper_open
recovery procedure: recheck_message, recovery_degree=1
open_gripper, recovery_degree=2
replace_gripper, recovery_degree=3
report, recovery_degree=4

And, information about "grasp" in this task list is enumerated in the task table and converted into grammatical rules stored in knowledge base as follows:

Task table

grasp t
* gripper_open
recheck_message 1
open_gripper 2
replace_gripper 3
report 4

Grammatical rules

PE→¬ (gripper_open) SR { recheck_message(1);
open_gripper(2);
replace_gripper(3);
report(4);
}

Conclusion

A methodology for knowledge representation for the error recovery module has been presented. The methodology can achieve the integration between error recovery techniques and control software through its formalization and simplicity. The error recovery module built by this process can be closely connected with the execution module. Knowledge representation is based on two types of errors and two attributes of tasks. Grammatical rules are used to create the knowledge base shared by the error preprocessor and the generator. The knowledge acquisition and reasoning procedures provide the ability required to handle error situation across different control levels.

References

1. Abu-Hamdan, M. & El-Gizawy, A. (1990). Recent Directions in Error Recovery Strategies for Flexible Assembly Systems, ASME Design Technical Conf. - 2nd Conf. in Flexible Assembly Systems, 1990, pp. 155-160.
2. Aldemo, A. & Andreasson, S. (1992). Models for Fault Tolerance in Manufacturing Systems, J. of Intelligent Manufacturing, Vol. 3, 1992, pp. 1-10.
3. Borchelt, R. & Alptekin, S. (1994). Error Recovery in Intelligent Robotics Workcells, International Journal of Production Research, 32(1), 1994, pp.65-73.
4. Chaar, J. K. (1990). A Methodology for Developing Real-Time Control Software for Efficient and Dependable Manufacturing Systems, Ph.D. dissertation, University of Michigan, 1990.
5. Freyermuth, B. (1992). Knowledge Based Incipient Fault Diagnosis of Industrial Robots, IFAC Fault Detection, Supervision and Safety for Technical Processes, IFAC Symposia Series No.6, 1992, pp.369-375.
6. Gini, M. & Gini, G. (1983). Towards Automated Error Recovery in Robot Programs, IJCAI, 1983, pp.821-823.

7. Hassapis, G., Petrou, L. & Kleftouris, D. (1987). A Computer Aided Shutdown System Analysis and Design Procedure, Computer in Industry, Vol.9, 1987, pp.115-125.
8. Jafari, M. (1990). Petri Net Based Shop Floor Controller and Recovery Analysis, IEEE Int. Conf. on Robotics & Automation, 1990, pp.532-537.
9. Kokawa, M., Miyazaki, S. & Shingai, S. (1983). Fault Location Using Digraph and Inverse Direction Search with Application, Automatica, 19(6), 1983, pp.729-735.
10. Kumaradjaja, R., DiCesare,F. & Goldborg, G. (1989). A Schema-based Representation for Execution Monitoring and Error Recovery in Assembly Systems, IEEE Int. Symposium on Intelligent Control, 1989, pp.33-37.
11. Lee, M., Hardy, N. & Barnes, D. (1983). Error Recovery in Robot Applications, Proc. of the 6th British Robot Association Annual Conf., 1983, pp.217-222.
12. Moon, Y. & Moodie, C. (1989). A Framework for Failure Recovery in a Manufacturing Cell, Int. J. of Advanced Manufacturing Technology, Vol.4, 1989, pp.144-156.
13. Narayanan, N. & Viswanadham, N. (1987). A Methodology for Knowledge Acquisition and Reasoning in Failure Analysis of Systems, IEEE Trans. on Systems, Man & Cybernetics, 17(2), 1987, pp. 274-288.
14. Smith, R. & Gini, M. (1992). Error Management for Robot Programming, Journal of Intelligent Manufacturing, 3, 1992, pp.59-73.
15. Srinivas, S. (1977). Error Recovery in Robot Systems, Ph.D. dissertation, California Institute of Technology, 1977.
16. Tam, K. Y. (1989). Linguistic Modeling of Flexible Manufacturing Systems, Journal of Manufacturing Systems, 8(2), 1989, pp.127-137.
17. Tsukada, T. & Shin, K. G. (1993). Intelligent Disruption Recovery for Decentralized Manufacturing Systems, IEEE International Conference on Robotics and Automation, 1993, pp.852-857.
18. Wu, H. J. & Joshi, S. B. (1995). A Methodology for Knowledge Representation of an Execution Error Recovery Module for Shop Floor

Control, working paper, Department of Industrial Design, Tung-Hai University, 1995.

製造現場控制中錯誤補正模矩之知識表達

吳銜容*

摘　　要

製造系統中之錯誤補正通常被視爲獨立活動，錯誤補正與其它活動在控制系統中的整合行爲因而被忽略。欲整合控制軟體與錯誤補正技術，必須明確的表達與錯誤補正相關的知識。本文嘗試提出一知識表達方法用於製造現場控制之錯誤補正模矩。其主要目的爲建立控制軟體與錯誤補正模矩間之連接。經由執行模矩與錯誤補正模矩的連結，此知識表達方法可通用於階層式製造系統中之不同層次。

*東海大學工業設計學系