An exception-handling framework

Visser, A.

Published in:
International Journal of Computer Integrated Manufacturing

DOI:
10.1080/09511929508944645

An exception-handling framework

A. Visser
Department of Computer systems
University of Amsterdam
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
arnoud@fwi.uva.nl

The exception-handling framework described in this paper creates an opening to compare different exception-handling approaches in a structured way. This comparison is made, linking taxonomies of different research-groups together. Concurrently the framework specifies a general data-structure to store knowledge about exception-handling, which makes it easier to adapt the proposed taxonomy in the implementation of existing and impending work-cell controllers and production planners.

Introduction

The manufacturing of products is a process that requires the knowledge of a large spectrum of area's. For each area a rich set of concepts, tools and taxonomies is developed to facilitate the finding of optimum solutions. To plan the manufacturing process of a (new) product enough understanding is needed on each area to be able to judge the relevance of the area for its specific application. An attempt has to be made in the manufacturing community to come to the definition of a common information model, to facilitate the communication between experts in different planning area's. Without a unified and unambiguous understanding of terminology easy partitioning of the planning tasks and easy interfacing between the planning groups will become very difficult.

This paper is an attempt to give a survey of the taxonomy used by exception handling experts, leading to a proposal on a common taxonomy about exception handling. The taxonomy described here is developed in the CIM-PLATO[1] and the IRAS project[2]. IRAS is an ESA funded project to define and demonstrate a control concept for task oriented operations with interactive adjustment of operation sequences and parameters. CIM-PLATO is an ESPRIT funded project to develop and demonstrate an environment to configure and connect a set CIM planning tools. These tools are for instance programs to plan the configuration of flexible manufacturing and assembly systems, to schedule the assignment of operations to those systems and to generate execution programs for the different systems. In both applications there is a clear need for techniques to increase the fault-tolerance of robots [15]. For manufactures it is important to that the robot is highly available, for the space community it is important that the robot is highly reliable.

To summarise the meanings and relations of the concepts revealed in this article; they will be described in a data-structure. This approach has several advantages. Relations with other concepts can be represented in a structured way, which makes it possible to concentrate on the definition of properties of the concept itself. Definition of a concept as the cluster of other concepts is often convenient, a process that is formalised by defining memberships. An other advantage is that the use of the taxonomy is encouraged, because a critical design step to implementation is already performed. As format of the description we have chosen EXPRESS[3], a forthcoming standard for data-model specification defined by the ISO. For easier understanding we illustrate the description with figures of the EXPRESS-G format. Those illustrations are complimentary, and are meant to show the relations between the entities. Some details, although described in the textual representation, are hidden in the

graphical representation. This is done for clarity; the graphics are meant for survey, not to give a complete description.

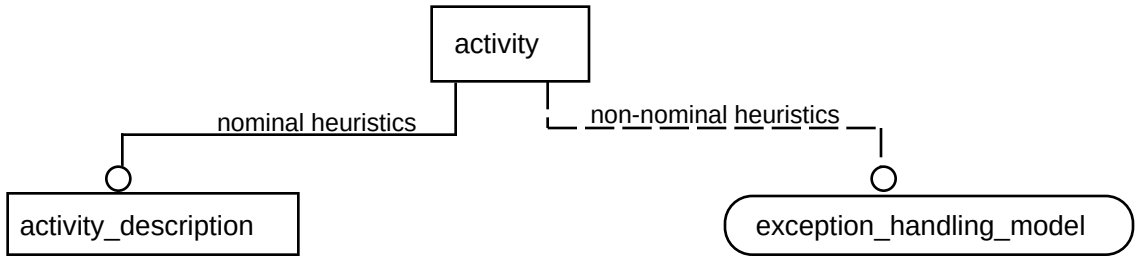
The activity to be performed

Think of a production area, with an automated resource instructed to perform certain activities. For instance a robot that has to assemble a mainframe computer by linking CPU units, in a unique and customised configuration. This job is worked out in a CIM-environment in a sequence of activities to be executed by the robot. Though sophisticated robot controllers can be instructed on a high level by commands as "Pick Up CPU-231" or "Grasp Gripper-Fixture of CPU-231"[2], still the majority of the controllers can only understand simple commands as "Move to pointB". All possible commands for robot-controllers shall be called in this article activity. The activity will naturally have a description depending on the robot-controller.

We will concentrate on the taxonomy of the things you want to known if something goes wrong with the execution of the activity (non-nominal heuristics). This taxonomy makes it possible to describe how the situation can be recognised non-nominal, to describe how a more detailed understanding of the situation can be obtained, to define possible sources of this exceptional situation, and to indicate possible activities to bring the situation back to nominal. The formulations can be hardcoded into the robot-controllers, but a more flexible approach is to create a database with this information. All the information together is called the exception handling model of the activity.

The need for an exception-handling model for each activity, instead of one for the whole sequence, was indicated by Srinivas[4]. Meijer [5] even indicated differences in monitoring and classification in three phases of the execution of the activity: the pre-, during- and post-phase. In the pre-phase the critical parameters of the activity are checked, before the operation causes any damage. This sort of exceptions is caused by information errors, due to a discrepancy between the internal model description of the environment and the description derived from sensor information. During the execution of the activity one concentrates on the motions of the robot. The exceptions that might occur are primarily caused by the motion of the robot itself. These are called operation errors; unanticipated changes in the physical environment caused by the robot operations. In the post-phase primary checks are made to be sure that the operation was successfully executed. It can take in account the final values of the nominal feedback, but also depend on information especially acquired for monitoring reasons.

So the information model on the activity that has to be performed, can be divided into two parts. On one side the information that is needed for nominal operation, on the other side the information that is need for non-nominal situations. This partition can be represented in the EXPRESS format in the following manner:




```
ENTITY activity
        -- an instruction for an active resource that is either directly executable or can be
        -- translated into executable commands of the target resource
        nominal_heuristics:                     activity description;
        non_nominal_heuristics:         OPTIONAL exception_handling_model;
END_ENTITY;

ENTITY activity_description
        -- Information needed during nominal control, depending on the sort of controller.
        -- illustrations of this information are the type of activity (GRASP), the workpiece
        -- where the activity works on (CPU 231) or the position the robot has to go to
        -- (pointB or [2.3,4.5,6.7]).
END_ENTITY;
```

The entity exception_handling_model will be described in the following paragraph. This is indicated in the express-G format by the round edges of the box.

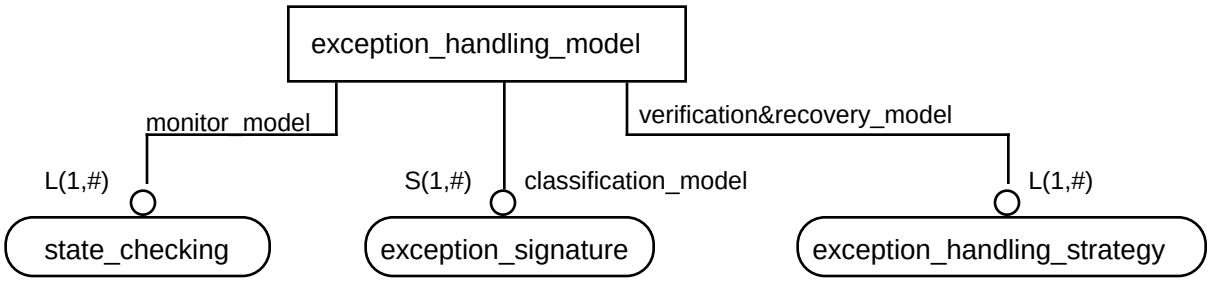
The exception handling model

An exception_handling_model represents the information available for a certain execution-phase of the activity. The model can be partitioned in three submodels. In the monitor model a description can be made which activities have to be done, to distinguish the nominal situation from non-nominal. In the classification model information about the relations between observed features, the sort of exception and possible reasons for that exception can be stored. Available (sensor) activities that can clear still existing uncertainties are saved in the verification & recovery model, together with the activities that have the potentials to recover the situation in the work-cell to normal. No separation between verification and recovery is made because a successful recovery often generates a lot of information about what was exceptional in the state before, while a successful inspection of critical features facilitates the recovery of the system. Complex search activities are often difficult to classify as a pure verification or pure recovery activity.

The partition of the exception handling model in submodels is not similar for different research groups. Dependent on their interest and application a research group works out some stages of the exception handling in more detail, while other stages could be assumed to be simple. Although the partitioning is not similar for different researchers, there are entities that always show frequently up in the submodels In the following paragraphs an attempt is made to indicate those entities.

The partitioning presented in this paper can be justified in the following way. Dependent on the constraints of the application you can locate the models at different devices. In the monitor model information is stored that is needed during the utilisation of the robot. This model has to be loaded on the real-time robot controller. All other information can be made available later; i.e. the data can be stored outside the real-time environment. The classification can for instance be done totally off-line; the robot can be used for other

things. The record of acquired sensor values is only thing needed during analysis. Verification & recovery is an iterative process, with every time a planning step and an execution step. Although the planning can take place on an other device, from time to time the robot is needed. The implementation constraints are clearly different for the three models, what justifies the separation made.




```
ENTITY exception_handling_model
        -- All information about possible exceptional situations that is specific for the execution
        -- phase of the activity.
        monitor_model:                          LIST(1,#) OF state_checking;
        classification_model:                   SET(1,#) OF exception_signature;
        verification&recovery_model:    LIST(0,#) OF exception_handling_strategy;
END_ENTITY;
```

Meijer[5] makes a separation into a monitor-, diagnosis- and exception_handling-area. In the diagnosis area distinction is made between identification & classification of the failure, and the process of updating & verification of the environment model. As exception handling plans mixtures of heuristics, planning functions and activities are indicated.

In the (H)ERA project [6] four areas are indicated: monitoring, symptom detection, diagnosis and recovery. Here the monitoring is just the gathering of information for post-mission performance characterisation. Checking if the information is inside specific boundaries, is defined as part of the symptom detection functions. In most cases an immediate reaction of the robot after the detection of an exception is programmed. This reaction must create a safe situation, to gain time for diagnosis and recovery. For the diagnosis an analysis of recorded data is made, new sensor requests are not anticipated. Recovery is done by sending a new plan to the space robot.

The state checking information

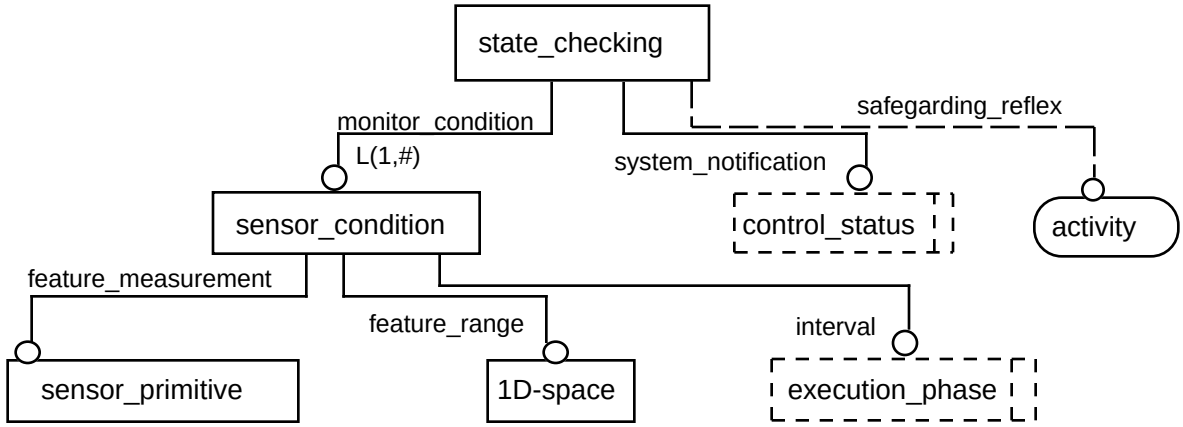
The state_checking information represents the instructions about the acquisition of physical properties and system internal states, the definition of boundaries between nominal and non-nominal feedback, the message that has to go to the system if an anomaly is found, and the activity to be undertaken to bring the system to a for this moment safe situation.

The purpose of sensor requests doesn't have to be pure exception based. Some information is also interesting for quality control, or give general state information for a supervisor ('where is the robot?'). In general the requests are accomplished by sensor operations, processes that represent the capabilities of the sensing system. The acquired data can be stored for later analysis. Sensor operations can make use of information gathered by other sensor operations to fulfil their task. For instance the sensor operation 'Wall Sensor' of a mobile robot with sonar can update the global map with the lines in the local sonar map if it knows the current position of the robot[8]. The behaviour of the sensor operations can be adapted to the environment by giving parameters to the routine. One can think of an

estimation of the correctness of the old global map, or the estimated reflection-coefficient of the wall.

For exception handling it is important that a clear but flexible separation between a nominal and non-nominal state is made. Here we will use a special class of the sensor operations: the sensor primitives. Sensor primitives are sensor operations that measure a single, but often critical, feature of the environment. The definition of boundaries for the measured feature will create two 1-dimensional feature spaces, one defining the nominal feedback, the other the non-nominal feedback. In the example of the mobile robot we can think of a sensor primitive that returns the distance to wall (in meters), what can be classified as non-nominal for the range < 0.5. This choice means not those sensor operations that return more dimensional features cannot be used. It means that one has to combine this sort of operations with other sensor operations in such a way that a single number is returned. A sensor that returns the position and orientation of an obstacle gives valuable information, which can be used during the classification and replanning phase, but for the monitor the time to contact is the important feature, and that number can be calculated if one combines this information with the current position and velocity of the robot. The advantage of restricting the boundaries to 1-D space is that these can be easily represented in a data-structure, in contrast to more dimensional spaces [14]. All the complex and application dependent combinatory functionality is in the sensor primitives, and doesn't need to be stored in an object oriented database.

At what phase the sensor primitive has to be executed is defined in the interval-slot. After the detection of an exception, the monitor system has to know how to notify the rest of the system about this situation. The message will depending on the robot controller, but one can think about types of messages as (OK, ERROR, WARNING) or ('nominal', 'non-nominal'). A good preventive warning system, followed by appropriate actions, can save many problems. The safegarding_reflex will in most case's be an emergency-stop, but in some case's activities as retract or drop can be preferred.




```
ENTITY state_checking
        -- Guarding of the operation state of the production resource by checking if the sensor feedback lay
        -- within pre-defined intervals
        monitor_condition:                  LIST(0,#) OF sensor_condition;
        system_notification:                          control_status;
        safegarding_reflex:                  OPTIONAL activity;
END_ENTITY;
```

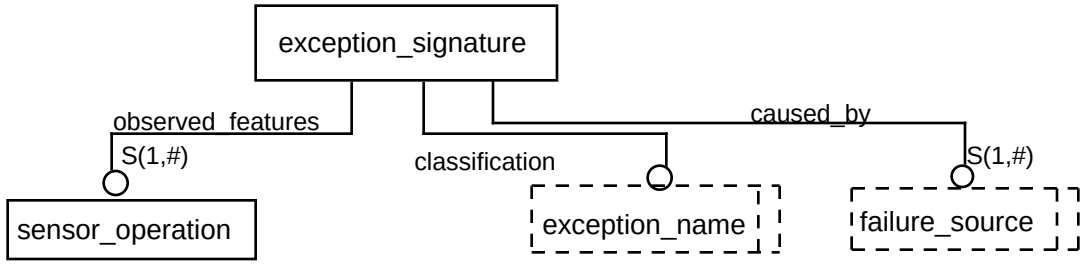
```
ENTITY sensor_condition
        -- A constraint imposed on the measurement of a single feature during a certain interval
        feature_measurement:                                    sensor_primitive;
        feature_range:                                                  1D-space;
        interval:                                                execution_phase;
END_ENTITY;

TYPE execution_phase = ENUMERATION OF
        (PRE, DURING, POST);
END_TYPE;

TYPE control_status =
        -- An immediate indication or notification of the occurrence of a severe problem
        -- (malfunction) or abnormal condition within the control system. The precise
        -- format depends on the interface to the other level controllers.
END_TYPE;

ENTITY sensor_primitive SUBTYPE OF sensor_operation
        -- A sensor primitive represents the measurement of a single feature from the
        -- environment. The sensor primitive is related to an actual sensor through a set of
        -- processing layers, which can include other sensor_primitives or sensor_operations.
END_ENTITY;

ENTITY 1D-space
        -- One dimensional feature space. This can be an infinite number of points or
        -- line-segments, in the format of the measured feature
        Limited_by:                                          SET[1,#] of boundary;
END_ENTITY;

ENTITY boundary
        -- limitation of an area, or an included or excluded point
        operator:                       ENUMERATION OF (relational, equality);
                                                        -- for instance <=, > or !=
        number:                         ENUMERATION OF (INTEGER, REAL);
                                                        -- units of measured feature
END_ENTITY;
```

By offering a slot for the different execution phases of the activity, we can represent the different timing approaches on exception handling taken by diverse research groups. The approach of researchers who consider heuristics valid for the whole sequence of activities [9] can be represented by defining a parent activity that represents the sequence. The exception_handling_model for the parent are valid for all descendant activities. So for the descendant activities the optional slot 'exception_handling_model' is not used. The approach of Srinivas, defining heuristics for every activity, can be described by filling in DURING in the execution_phase slot. Doyle's approach[13], an explicit representation of pre- and post-conditions of activities, which are checked by verification operators (logical sensors) can be represented with PRE and POST, while this time the DURING is not used. To describe the models defined in [5] we need all three possibilities.

The exception signature

The classification model is a set of exception signatures, each representing a certain classification of the situation, and assumptions that can be made in such situation. An example of such a classification is 'Collision' or 'Handled Object lost'. For each classification An assumption can be made on forehand about how this situation will manifest itself in distinctive values of characteristics features. This information is stored in

the exception_signature, together with several causes that could explain the occurrence of the exception.



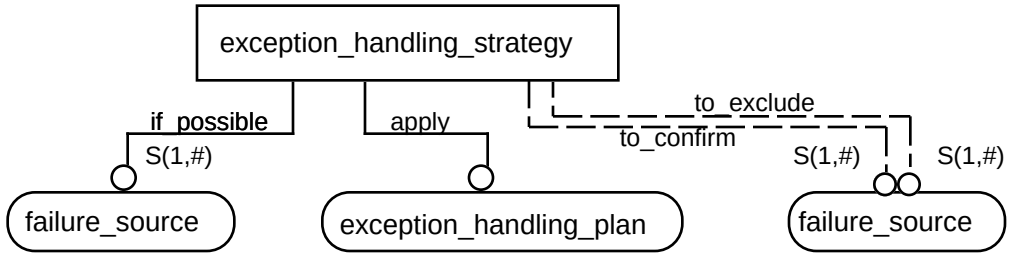

```
ENTITY exception_signature
        -- an exception signature represents the information necessary to classify the non-
        -- nominal situation in more detail then just the indication that something is wrong,
        -- by indicating the relation between observed features, a classification and several
        -- sources of unexpected events.
        observed_features:                  SET(1,#) OF sensor_operation;
        classification:                                 exception_name;
        caused_by:                                SET(1,#) OF failure;
END_ENTITY;

ENTITY sensor_operation
        -- A sensor primitive represents the measurement of features from the
        -- environment. The sensor operation is related to an actual sensor through a set of
        -- processing layers. The operation is a request for the functionality of sensor
        -- input, hiding the all but trivial data processing at this level. ref. [7, 8]
END_ENTITY;

TYPE exception_name = ENUMERATION OF
        (Collision,
        Handled Object Lost,
         ...);
END_TYPE;

TYPE failure_source = ENUMERATION OF
        (Accuracy Loss Robot,
        Deformed Handled Object,
        New Obstacle,
        Vibrations,
        ...);
END_TYPE;
```

Srinivas[4] did initial work on the definition of the relations between failure reasons and exceptions. The relation was not explicitly used, but was implicitly used in the definition of a discrimination network of diagnostic activities. Chang et al [10] made an explicit association between expected features and failures. For space projects the isolation of the source of the exception is an important issue. Much of the diagnosis effort is put in this identification process. Meijer[5] also considered different causes of an exception, by defining different variants of an exception-classification. Examples of these variants are 'Collision with unknown object' or 'Collision with object O at position P'.

The exception handling strategy

As soon the robot control system realises that the situation is non-nominal, choices have to be made which activities are adequate for the given situation. Sometimes not enough

information about the current situation is available, so strategies to acquire new knowledge can be followed. Sometimes enough other work is still available for the robot. A rescheduling of the plan is at that moment a good strategy. Alternatives, already considered during the preparation, can be tried. A part of the original plan is then replaced in a way defined in an replanning strategy. Each strategy is assumed to be effective for a limited set of failures. A successful execution of a strategy can give clues about the cause of the exception. This can be indicate by a set of excluded and confirmed failure sources.

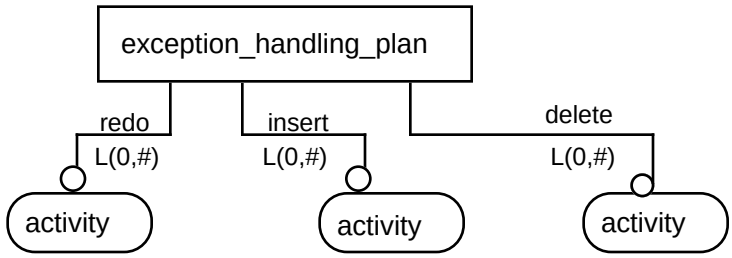



```
ENTITY exception_handling_strategy
        -- the guide-lines on how an exception has to be handled, depending plausibility of different
        -- causes of the exception
        if_possible:                          SET(1,#) OF failure_source;
        apply:                                  exception_handling_plan;
        to_confirm:                 OPTIONAL SET(1,#) OF failure_source;
        to_exclude:                 OPTIONAL SET(1,#) OF failure_source;
END_ENTITY;
```

Attachment of sensor activities to each combination of robot activity and failure is a common approach. López-Mellado et al. [11] presented for instance a robot in work-cell, characterised by a set of sites. Such a site could contain objects. In the case of exception, the robot detects for instance against its expectation no object on a site, a collection of test on the occupancy of different sites is performed. The rules to build up such collection, can also be represented as a collection of failure's reasons as 'Object on siteX' linked with plans as 'Test if siteX is occupied', which confirms the mentioned reason. Meijer indicated networks of sensor requests to gain enough information to be able to select a recovery strategy. This approach can be represented by initially consider all possible reasons, to trigger diagnostic activities as 'Identify Object' on conflicting reasons as 'Collision with unknown object' and 'Collision with object O'. The choice between recovery strategies 'PlanMoveOver' or 'PlanPathPassingO' can be indicated by a trigger on only the first reasons for 'PlanMoveOver', and only the last reason for the other recovery plan.

The exception handling plan

An exception handling plan must indicate what is a reasonable think to do in the current non-nominal situation. It means that the plan has to indicate new choices for the planning functions that are able to generate a new or modified sequence of actions. Unfortunately, general planner systems suffer from a complexity problem. A general way to instruct such a general planner system is difficult. The approach taken in IRAS[2] was to specify the plan with not complete initiated activities. The deficiencies indicate the need of new planning efforts. Activities of the original sequence, or special developed activities stored in a contingency library, can be used as sources for those plans. Old activities can be indicated to be redone or to be deleted, and new activities can be indicated to be inserted. Naturally, such a set of exception handling plans must be tailored for a specific application.




```
ENTITY exception_handling_plan
        -- a scenario of the adaptations that has to be made in the original plan, to gain knowledge
        -- of the status of the system and to bring it to a state were it can continue.
        redo:                                   LIST(0,#) OF activity;
        insert:                                 LIST(0,#) OF activity;
        delete:                                 LIST(0,#) OF activity;
END_ENTITY;
```

Meijer[5] indicated recovery plans as for instance 'PlanPathPassingO', defined as the initiating of a path planner, notified of the existence of object O, followed by retry of the parent activity with this new path. This plan can be represented in our model as pure redo of the parent activity, but an empty slot for the path in activity_description. Gini[12] also reports that a recovery module can make use of the current activity, but needs to do that on more than one level of abstraction. This can be represented by indicating a redo on the highest level of abstraction, and indicating on lower levels the parameters in the activity_description that have to be forgotten.

Conclusion

In this paper an exception handling framework is described that can represent many approaches taken in exception handling. This framework has slowly be grown in work for different international projects. The need to explain and configure the work to many different partners made it a complete and detailed description of what's possible in exception handling in a structured environment as an industrial shopfloor. The concepts used are defined in a mature way, together with their internal relations, leading to a proposal for a taxonomy in the exception handling field.

References

1 Welz, B.G., L. Camerinha, T.C. Lueth, S. Münch, L. Stocchiero, J. Tramu and A. Visser "A Toolbox of Integrated Planning Tools - A Case Study" in the Proceedings of the workshop on "Interfaces in industrial systems for production and engineering", IGD, March 1992, Darmstadt, Germany

2 Foth, P. "Interactive Remote Automation&Remote Servicing - Final Report" ESA Study 9459/91/NL/JG, May 1993, Bremen

3 "EXPRESS Language Reference Manual", ISO document 184/SC4/N466, 1990

4 Srinivas, S. "Error recovery in robots through failure reason analysis" AFIP Proceedings of National Computer Conference, Anaheim, C.A. 1978

5 Meijer, G.R. "Autonomous shopfloor systems - a study for exceptin handling for robot control" PhD. thesis, Amsterdam, 1991

6 Elfving, A. "IRAS Non-nominal feedback" Private communication, Noordwijk, Jan 1993

7 Weller, G.A. F.C. Groen and L.O. Herzberger "A sensor processing model incorporating error detection and recovery" In Proceedings of the NATO International Advanced Research Workshop on Traditional and Non-traditional Sensing, Edited by T.C. Henderson, 351, NATO-ASI Series F63, Springer-Verlag, Berlin Heidelberg, 1990

8 Boer, G.A. den, G.D. van Albada, L.O. Hertzberger, G.R. Meijer, J.-B. Thevenon, P. LePage, E.J. Gaussens, F. Arlabosse, "An Exception Handling Model Applied to Autonomous Mobile Robots" Proceedings of Conference Intelligent Autonomous Systems-3, Pittsburg, 1993

9 Isermann, R. "Process fault detection based on modeling and estimation methods - A survey" Automatica 20(4): 387-404, 1985

10 Chang, S.J., F. DiCesare and G. Goldbogen "Evaluation of diagnostisablilty of failure knowledge in manufacturing systems" Proceedings of IEEE conference on robotics and automation. 1990

11 López-Mellado, E. and R. Alami "A failure recovery scheme for assembly workcells" Proceedings of IEEE conference on robotics and automation 1990

12 Gini, M. and G. Gini "Towards Automatic error recovery in robot programs" Proceedings of the 8th International joint conference on Artificial Intelligence, August 1983

13 Doyle, R. J., D.J. Atkinson and R.S. Doshi. "Generation perception requests and expectations to verify the execution of plans" Proceedings of AAAI 86 1986

14 Requicha, A.A.G. and H.B. Voelker: 'Solid Modeling: a historical summary and contemporary assessment' IEEE Computer Graphics and Applications, 2(2), pp. 9-24

15 Siewiorek, D.P. Architecture of fault tolerant computers IEEE Computer, vol 17, no. 8 August 1984.