Towards Intelligent Execution Supervision for Flexible Assembly Systems

L. Seabra Lopes and **L.M. Camarinha-Matos**

Departamento de Engenharia Electrotécnica
Universidade Nova de Lisboa
Quinta da Torre, P-2825 Monte da Caparica, Portugal

ABSTRACT

Research results concerning error detection and recovery in robotized assembly systems, key components of flexible manufacturing systems, are presented. The approach to the integration of services and the modelling of tasks, resources and enviroment is described. A planning strategy and domain knowledge for nominal plan execution and for error recovery is presented. A supervision architecture provides, at different levels of abstraction, functions for dispatching actions, monitoring their execution, and diagnosing and recovering from failures. Through the use of machine learning techniques, the supervision architecture will be given capabilities for improving its performance over time.

1. INTRODUCTION

The ability to produce highly customized products, in order to satisfy market niches and comply with the new challenges of a globalized economy, requires the introduction of new features in automation systems — *flexibility*, adaptability, versatility — relaxing cell structuration constraints and leading to the concept of Flexible Manufacturing Systems. The efficiency and economical success of such systems depend, however, on the capacity to handle unforeseen events, which occur in a greater number, due to the reduction in structuration constraints. The complexity of flexible manufacturing processes makes the supervision and maintenance tasks difficult to perform by humans. Therefore, in manufacturing systems, flexibility and autonomy are tightly related concepts. On-line decision making capabilities have to be included in supervision systems.

We have previously proposed an execution supervision architecture for flexible assembly systems having the following main functionalities [1,2]:

Dispatching — dispatch actions to the executing agents, driven by the scheduled task plan and synchronize the agents activities.

Monitoring — detect off-nominal feedback in the system during plan execution (deviations from normal behavior).

Failure Diagnosis — it is called when the monitoring function detects a deviation; in general, it consists of four steps: failure confirmation, failure classification, failure explanation and status identification.

Failure Recovery — determine a recovery strategy to bring the execution to a nominal state.

In this paper, the main modules of an execution supervisor for a robotized assembly cell are presented. The assembly cell, the execution infrastructure and the modelling approach are initially presented. A planning strategy and the domain knowledge for nominal plan execution and for error recovery are described. One main problem is the acquisition of knowledge about the task and the environment to support monitoring, diagnosis and recovery. For this purpose, the use of machine learning techniques was investigated, in the context of the European ESPRIT project B-LEARN II. Particular attention was given to the inductive generation of structured classification knowledge for diagnosis.

2. INTEGRATION AND MODELLING

Execution Infrastructure

The setup being used in our experiments is a robotic assembly cell (Fig. 1), composed of an industrial SCARA robot, three robot grippers, magazine and corresponding tool exchange mechanism, two special purpose feeders and one fixture. As feedback information sources, several discrete information sensors were integrated into the cell. Since, the most frequent execution failures are expected to be those in which the robot arm is involved, including collisions, obstructions and handling failures, a force and torque sensor was also included.

From the supervision point of view, the main limitation of the robot controller language is that it does not provide guarded movements. Since communication via teach pendant (TP) is very fast, each motion command is decomposed into a series of increments, executed sequentially via TP, until some condition is verified. A server process emulating the teach pendant (TP Emulator) and runing on a dedicated PC, due to the tight communication cycle, was first developed. An adaptation layer, added to the TP Emulator, provides guarded movements, as well as robot commands normally available via TP and commands of other cell resources, that are actuated via the robot controller output ports. The TP Emulator also provides information about robot errors. The TP Emulator plus the adaptation layer is called Operational Server.

Execution monitoring takes place in another server process. As it is not easy to make acquisition of large quantities of sensorial data in Unix workstations, and, on the other side, concurrency in Unix affects the sensor sampling rate, this process, called Low-Level Monitor (LLM) is run in another dedicated PC, where it is quite simple and cheap to implant a data I/O board. An execution Supervisor, runing on a Unix workstation, cordinates the behavior of the assembly cell. The low-level monitor checks conditions during the execution of actions, as specified by the Supervisor, and is able to answer questions about the state of the system during the diagnosis phase.

0-7803-3280-6/96/$5.00

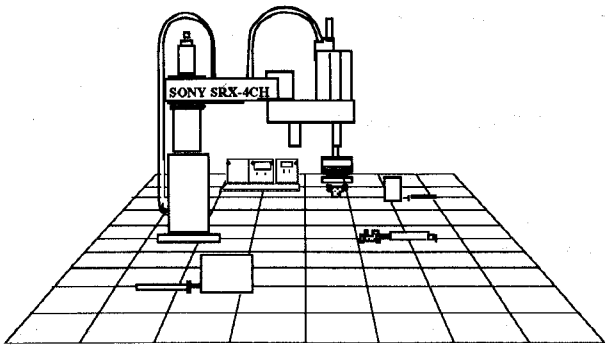

Fig. 1 — The Assembly Cell

Modelling Tasks and Environment

The execution supervisor contains the "intelligence" of the system, being responsible for all strategic decisions. The supervisor plans execution, defines conditions to be monitored, diagnoses and recovers from failures, and interacts with the human operator. Therefore, the supervisor needs models of the resources, the environment and the tasks to be performed. Independently of how these models are generated, a representation must be agreed upon in the first place.

The kernel of the supervisor is implemented in Prolog, a logic programming language, well known for its powerful term unification procedure, for its inference control strategy and for easy symbol manipulation. Additionally, in order to better structure the main concepts, an in-house developed frame engine, Golog [7], that runs as a shell in Prolog, is used. Golog provides the main features of the frame-based, object-oriented and reactive programming paradigms, namely relations with inclusive and exclusive inheritance, methods and several types of demons. This seems to be the right approach to model, not only the structural aspects, but also the dynamic properties of the assembly cell [2].

The basic structure that integrates de knowledge of the Supervisor is a taxonomy of entities related to the domain (see Fig. 2 for an outline of its current version). The assembly resources are devided in operational resources, sensorial resources and resource storage units. In the first group, robots, grippers, fixtures, feeders, pallets, etc., are considered. In our experimental setup, presence sensors and the force & torque sensor are examples of sensorial resources, while a tool magazine is an example of a resource storage unit. An adequate collection of assembly resources may constitute an assembly cell.

The assembly resources manipulate artifacts that can fall in three categories: assembly parts, assembled products and unexpected or unknown objects. Assembly parts are used as components of the assembled products. Notice, however, that the expression 'Assembly Component' is used, not to refer to the physical entity 'Assembly Part', but to a logical entity that aggregates all the information concerning a given component, namely the type of part that will represent this component, the mating referentials and the relation with other components. In general, a given type of assembly part can represent different components in the same product. Each component will correspond to a node in the assembly precedence graph and in the assembly contacts graph of a given product.

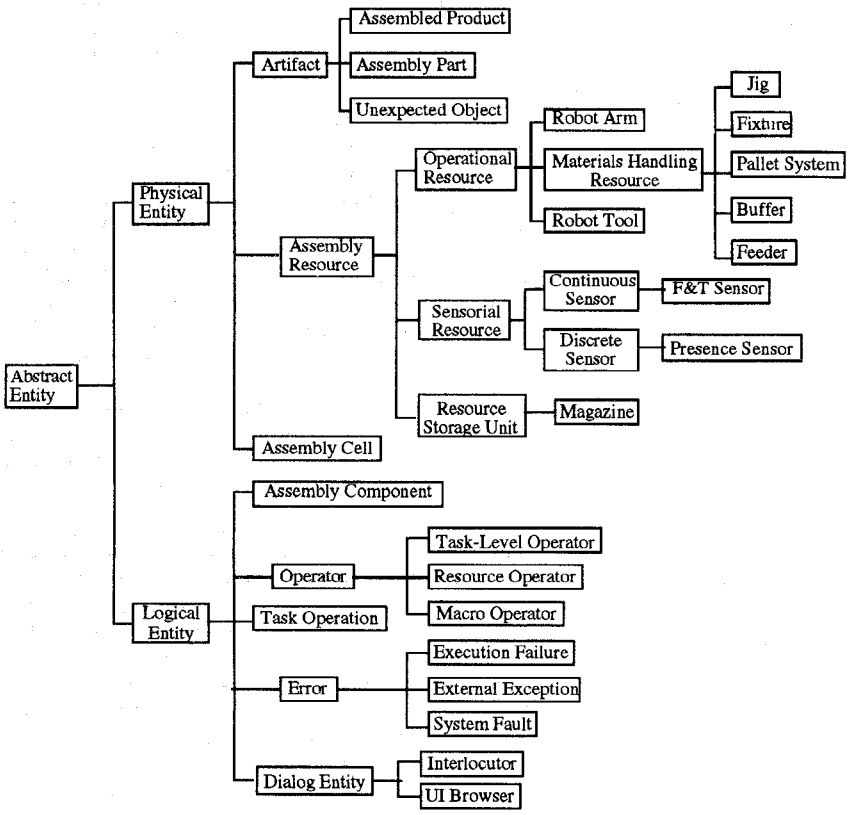

Fig. 2 — Taxonomy of Assembly Entities

For instance, consider the Cranfield benchmark, a well known laboratory product used for testing in the assembly domain. It is a pendulum composed of seventeen parts: two side plates, four spacer pegs, a shaft, a lever, a cross bar and eight locking pins. Physically, each locking pin belongs to the same category of part. However, in relation to the benchmark, each pin plays a specific role, having different precedences and contacts (see Figs. 3-4).

The functionalities of each assembly resource are specified by a list of operators of the category named 'Resource Operator'. These are the elementary operators or skills of the system. For instance `approach`, `peg_in_hole` and `transfer` are elementary operators of a robot while `feed_part` is an elementary operator of a feeder.

In the frame representing each `operator`, the method `op_implementation` specifies the desired functionality. Operators are assigned to the resources via the

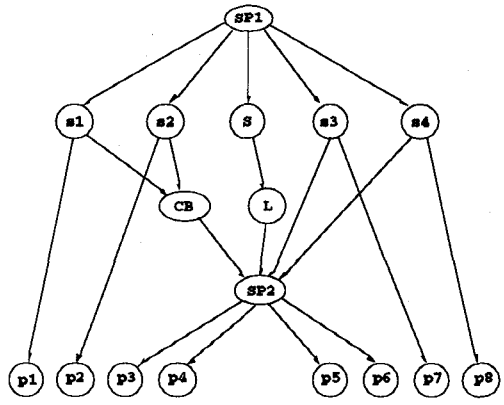

Fig. 4 — Cranfield benchmark: precendence graph

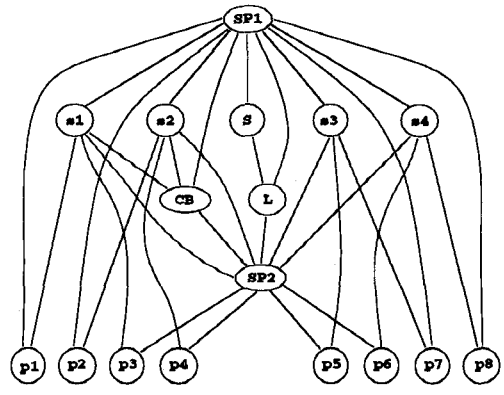

Fig. 4 — Cranfield benchmark: graph of contacts

relation `performed_by`. When this happens, the frame engine automatically establishes the inverse relation in the resources (in this case the `operators` relation). The frame engine allows that, from this point on, the methods that implement the operators can be called as if they belonged to the frames that represent the corresponding resources. For example, the method `transfer`, in a given robot R, can be called in the following way:

```
call_method(R,transfer^op_implementation,Loc)
```

Strategic decisions concerning future actions of the system are not taken based on the elementary operations. Namely, planning for nominal execution and for error recovery is performed at a higher level of abstraction, using the so called task-level operators. Examples of this category of operators are `pick_part_from_feeder`, `putdown`, `assemble_component`, `store_tool`, `feed_pallet`, etc., (14 operators in total). In each of these operators, its functionality, in terms of which properties are added or deleted from the state description, is specified in the slot `op_functionality`.

A generated task plan is a sequence of task operations, each of them represented by a frame in Golog. Each operation inherits from the corresponding operator the method `get_op_expansion` that calculates the sequence of elementary operations (i.e. calls to the resource operators) that are necessary to accomplish its goals. This is, basically a compilation or expansion procedure. For each task operation, the planning phase also generates the conditions to be monitored before the operation starts and the conditions that must be achieved after completion. A training phase may be necessary to determine the conditions to be monitored during the execution. These conditions constitute the monitoring profile of the operation.

In case the monitoring function detects some deviation between the actual state of the system and the expected conditions, the diagnosis function is called. Three categories of errors are considered [1]: execution failures are deviations of the state of the world from the expected state detected during the execution of actions; external exceptions are abnormal occurrences in the cell environment that may cause execution failures; system faults are abnormal occurrences in the cell resources hardware and software. The description of each error includes the procedure to identify it's occurrence based on sensorial manifestations.

Other entities without physical correspondence are, for instance, those related to the user interface, a very important module of a supervision system. Namely, each entity that can be visualized or affected by user interaction, will have one or more interlocutors, responsible for providing the right views of it in each situation. Finally, instances of 'UI Browser' represent active graphical windows and their contents in each moment.

3. PROGRAMMING BY DEMONSTRATION

In real execution, a feature extraction function is permanently acquiring monitoring features from the raw sensor data. The monitoring function compares these features with the nominal action behavior model. When a deviation is detected, the diagnosis function is called to verify if an execution failure occurred and, that being the case, determine a failure classification and explanation. For this function, additional features must be extracted. Diagnosis is a decision procedure that requires a sophisticated model of the task, the system and the environment. The final step, based on the failure characterization, is recovery planning.

The problem of building such sophisticated model is not easily solved. Even the best domain expert will have difficulty in specifying the necessary mappings between the available sensors on one side and the monitoring conditions, failure classifications, failure explanations and recovery strategies on the other. Also, a few less common errors will be forgotten.

The paradigm of Robot Programming by Demonstration (RPD) [4,10] seems indicated to overcome this type of difficulties. According to this paradigm, complex systems are programmed by showing examples of their desired behavior. In our approach, interaction with the human, seen as a tutor is fundamental. Usually, emphasis is put on robots learning from their own perception of how humans perform certain tasks. In our approach, RPD is broader and includes any interaction with the human that leads to improvement in future robot performance. Functions for training and learning are included in the supervisor architecture. An adequate user interface facilitates transfer of the human's knowledge to the system (Fig. 5).

The human will carry out an initial training phase for the nominal plan execution. The traces of all testable sensors will be collected during training in order to generate primitive skills and the corresponding monitoring know-

ledge. Also in the initial training phase, the human operator may decide to provoke typical errors, in order to collect raw data in error situations. Error classification knowledge is subsequently generated by induction. When a new failure is detected during real execution of the assembly system, the human operator is called to classify and explain that failure and to provide a recovery strategy for the situation. This is considered also as a training action, since the system history and the model of errors will be expanded and new knowledge will eventually be generated by incremental induction, therefore improving future system performance.

A qualitative approach to modeling errors was followed in this work. Depending on the available sensor information, a more or less detailed classification and explanation for the detected execution failure may be obtained. Therefore, the model of errors should be hierarchical or taxonomic. At each level of the taxonomy, cause-effect relations between different types of errors can be added.

The classification phase of the diagnosis task can be performed based on knowledge generated automatically by inductive learning. A new algorithm, SKIL (structured knowledge by inductive learning) [8,2], was developed to perform this task. SKIL generates hierarchies of anonymous concepts. Each concept is defined by a conjunction of attribute-value pairs. The number of specified attributes defines the abstraction level. The hierarchy is simultaneously a decision tree that, based on the values of the available discrimination features, can be used to recognize instances of the concepts. As far as diagnostic knowledge is concerned, the most important evaluation criterion is accuracy. Information on hierarchical problem decomposition, given to SKIL, leads to significant improvements in classification accuracy. Other methods to improve accuracy, that were investigated, were example pre-processing methods, namely example interpolation and feature construction [10]. Example interpolation, which consists of generating additional examples from "neighbor" examples, in order to achieve a better coverage of the input space, provided very good results, comparable to those obtained with hierarchical decomposition.

Typically, execution failures are caused by system faults, external exceptions or other past execution failures, although, in general, errors of the three kinds may cause each other [1]. Determining explanations for detected execution failures can become very complex, especially when errors propagate. Modeling errors in terms of taxonomic and causal links aims at handling this complexity [1,6].

Nevertheless, the crucial step in diagnosis seems to be training the system to understand the meaning of sensor values and learn a qualitative and hierarchical model of the behavior of each operation. Programming such model would be nearly impossible. Since the human defines the "words" (attribute names and values) used in the model, the human is capable of understanding the more or less detailed description that the model provides for each situation. It is then easier to hand-code explanations for the situations described in the model.

4. PLANNING AND FAILURE RECOVERY

When a given assembly task is selected for execution, the high-level specification contained in the process plan is finally instantiated. It is the moment to plan the execution, i.e. to determine all needed actions, their characteristics and parameters and their optimal sequence.

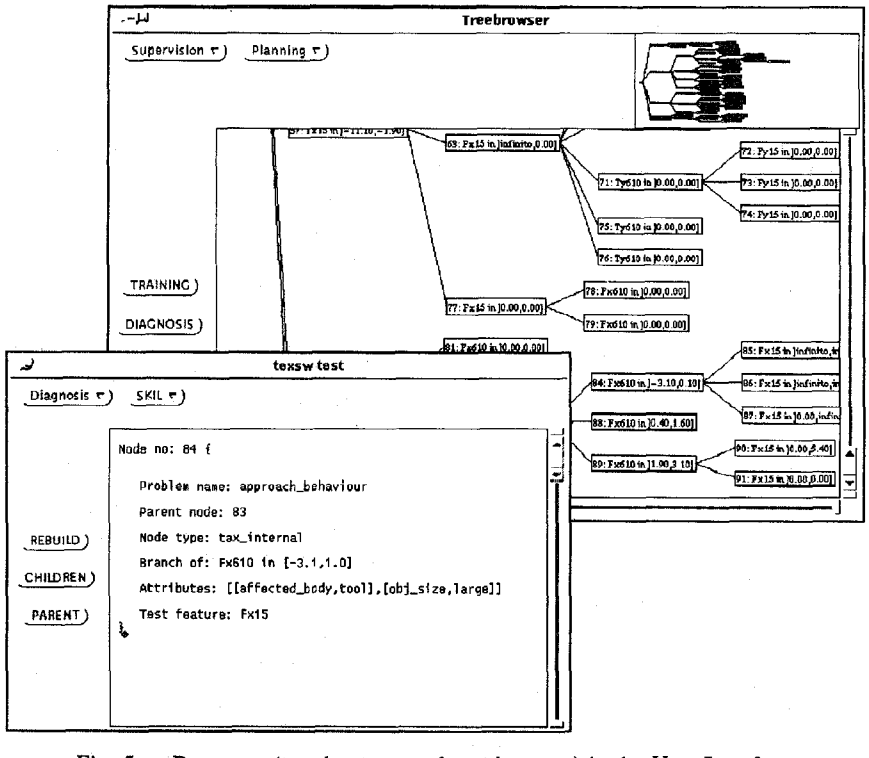

Fig. 5 — Browsers (tree browser and text browser) in the User Interface

For nominal planning, and also for failure recovery, a planner, in the AI sense of the term, was developed [9]. This planner, implemented in Prolog, uses a domain independent planning strategy, but takes into account domain knowledge provided in a pre-defined format. The planning strategy is, basically, a depth-first forward search procedure. From the initial state of the world, new states are generated, by applying operators of the considered domain, until the goal state is reached. In each step, a set of legal operator instantiations is determined, and evaluated according to domain dependent heuristics. The operator with highest score is selected for continuing the search. The way the planner uses the provided domain knowledge to select operators makes it a non-linear planner, since it can handle interaction between goals.

The advantage of the depth-first approach is that, when the heuristics are good, a solution is reached very fast. When that is not the case, the planner may have to spend a lot of time exploring uninteresting alternatives.

A new planning strategy, inspired in the well known A* algorithm (best-first search in a problem graph), was implemented. Like the previous stragegy, this one is also domain independent. In this case, domain knowledge is specified in three ways. First, the frame-based descriptions of the task-level operators (see example in Fig. 6) must be provided. To guide and control search, rules to estimate the cost of reaching the goal state starting from a given current state and rules to determine if two planning states are equivalent are required.

```
?- show_frame(assemble_component).

Frame: assemble_component {
isa: tasklevel_operator
op_cost: 1
op_functionality: [
  assemble_component
     (R,T,Obj,Comp,Part,Prod,Fix),
  [% Info:
     object_type(Obj,Part),
     part_tool(Part,T),
     component_contacts(Comp,LComp),
     mate(Prod,Comp,Part,_,Prec,_)],
  [% Keep-PC:
     current_tool(R,T),
     not(assembled(Comp,Prod,Fix)),
     fixture_with_product(Fix,Prod),
     not(robot_arm_breakdown(R)),
     not(tool_breakdown(T)),
     not(defective(Obj)),
     all(C,Prec,assembled(C,Prod,Fix)),
     all(C,LComp,
       [ assembled(C,Prod,Fix)),
         represented_by(C,X)]
         -> [not(defective(X))]) ],
  [% Del-PC:
     object_in_robot(Obj,R) ],
  [% Add-C:
     assembled(Comp,Prod,Fix),
     represented_by(Comp,Obj),
     robot_free(R) ] ]
get_op_expansion: get_assemb_expansion_mth
}
Yes

?-
```

Fig. 6 — Planning knowledge: a task-level operator

It is not possible to completely describe here a realistic planning example. Just for illustration, consider the Cranfield benchmark, already mentioned, and its seventeen parts. In our experimental setup we have special purpose feeders for side plates and cross bars. The locking pins and the spacer pegs are fed to the system in a pallet and the lever and the shaft in another pallet. Three different tools must be used. Running the planner with the incorporated domain knowledge on this problem finds an optimal plan — 53 operations — in about five minutes (see example of a generated task operation in Fig. 7). Considering that the average branching factor is 9, this result is acceptable.

The initial execution plan is generated for the nominal conditions. The representation of operators contains several assumptions. For instance, `not(defective(Obj))` is one of the pre-conditions of the `assemble` operator. If these assumptions are violated, execution failures will eventually occur. Often, the available sensor information is not enough to assert, with significant certainty, which is the state of the system. Even if the failure classification is determined with confidence, several failure explanations may still be possible. Imagine that, in executing the assembly plan of the Cranfield Benchmark, mating one of the spacer pegs with the side plate fails. Various explanations are possible: the spacer peg might be defective, the side plate might be defective, some unexpected object originating in the environment might be obstructing the mate operation, etc. Unless some sophisticated sensorial feedback is available, the supervision system will have to replan based on assumptions or beliefs. The recovery actions will be, simultaneaously, verification actions.

It should be noted that the planning strategies described above, show limited success in recovery planning, due to the difficulty in programming heuristics for all possible failure situations [9]. The programming by demonstration paradigm is therefore extended to address failure recovery. In this case, it is expected that the human operator will provide a failure recovery plan whenever the system cannot find one automatically. If this plan is sucessful, the error recovery episode (described by the failed operation, the failure diagnosis and the recovery plan) is stored in a form that preserves the essential information but not the details.

In our system, to achieve this representation, abstraction and deductive generalization mechanisms are applied [11]. The generalization procedure takes inspiration in the early STRIPS/PLANEX planning and execution system [3] and in Explanation-Based Learning (EBL) [5]. Generalization, in this case, is the deduction of an error recovery episode that, if adequately instantiated with constants, becomes the original episode. Therefore, each constant in the original episode, wherever it appears playing the same role, must be represented by one and only one variable in the generalized episode.

Unfortunately, the knowledge produced by deductive generalization is often too specific and, therefore, expensive both in terms of storage space and retrieval time. Applying abstraction mechanisms in order to reduce the level of detail in an episode description seems to be the right approach. In

particular, the use of a taxonomy of operators as the reference structure enables to ignore some parameters and some operations. The abstract operators have no consistent definition, and so the abstract plan cannot be proved. It can, however, be used as a guide in replanning.

In future system performance, when similar situations arise, these episodes are indexed, using information on the failed operation and on the failure diagnosis, and retrieved, and the recovery strategies that were found successful in the past are adapted for the new situations. Preliminary results obtained with this approach are reported in [11].

5. CONCLUSIONS

On-line decision making capabilities must be included in manufacturing systems in order to comply with the new requirements of flexibility and autonomy. Research results concerning error detection and recovery in flexible assembly systems were presented. The proposed supervision architecture includes functions for dispatching of actions, execution monitoring and failure diagnosis and recovery. An effort is being made to integrate different techniques and programming paradigms in order to achieve something that really works. This integration requires that the structure, representation and organization of the main concepts to be clearly defined.

The lack of comprehensive monitoring and diagnosis knowledge in the assembly domain suggests the use of the robot programming by demonstration paradigm. The general approach is to collect examples of normal and abnormal behavior of each operation or operation-type/operator and generate a behavior model that the diagnosis function will use to verify the existence of failures, to classify and explain them and to update the world model.

Developments in planning for nominal execution and for error recovery were presented. The used planning strategy is forward chaining and based on the A* search algorithm. Current research is focussing on the learning aspects in recovery planning.

```
?- show_frame(assemble_cb1).

Frame: assemble_cb1 {
isa: task_operation
in_plan: cranfield_plan
operator: assemble_component
parameters:
  [sony,gp2,o#21,cb1,cross_bar,cranfield,fix1]
preconditions: [
     current_tool(sony,gp2),
     not(assembled(cb1,cranfield,fix1)),
     fixture_with_product(fix1,cranfield),
     not(robot_arm_breakdown(sony)),
     not(tool_breakdown(gp2)),
     not(defective(o#21)),
     assembled(peg1,cranfield,fix1),
     assembled(peg2,cranfield,fix1),
     not(defective(o#15)), % peg1
     not(defective(o#14)), % peg2
     not(defective(o#6)),  % sp1
     object_in_robot(o#21,sony) ]
goals: [
     assembled(cb1,cranfield,fix1),
     represented_by(cb1,o#21),
     robot_free(sony) ]
monitoring_profile: % only after training
next_operation: assemble_sft1
}
Yes

?-
```

Fig. 7 — A Task Operation

Acknowledgements

This work has been funded in part by the European Community (Esprit projects B-Learn II and FlexSys) and JNICT (project CIM-CASE).

References

[1] Camarinha-Matos, L.M., L. Seabra Lopes and J. Barata (1994) Execution Monitoring in Assembly with Learning Capabilities, *Proc. of IEEE Int'l Conf. on Robotics and Automation*, San Diego.

[2] Camarinha-Matos, L.M., L. Seabra Lopes and J. Barata (1996) Integration and Learning in Supervision of Flexible Assembly Systems, *IEEE Transactions on Robotics and Automation* , vol. 12, pp. 202-219.

[3] Fikes, R.E., P.E.Hart and N.J. Nilsson (1972) Learning and Executing Generalized Robot Plans, *Artificial Intelligence*, 3 (4), pp. 251-288.

[4] Kang, S.B. and K. Ikeuchi (1995) Toward Automatic Instruction from Perception. Temporal Segmentation of Tasks from Human Hand, *IEEE Transactions on Robotics and Automation*, vol. 11, pp. 799-822.

[5] Kedar-Cabelli, S.T. and L.T. McCarty (1987) Explanation-Based Generalization as Resolution Theorem Proving, *Proc. 4th Int'l Workshop on Machine Learning*, Irvine, CA, pp. 383-389.

[6] Mozetic, I. (1991) Hierarchical Model-based Diagnosis, *Int. J. of Man-Machine Studies*, vol. 35, pp. 329-362.

[7] Seabra Lopes, L. (1994) *Golog 2.0. Um Gestor de Objectos em Prolog* (in Portuguese), Technical Report, UNL-12-94.

[8] Seabra Lopes, L. and L.M. Camarinha-Matos (1995) Inductive Generation of Diagnostic Knowledge for Autonomous Assembly, *Proc. IEEE Int'l Conf. on Robotics and Automation* , Nagoya, Japan.

[9] Seabra Lopes, L. and L.M. Camarinha-Matos (1995) A Machine Learning Approach to Error Detection and Recovery in Assembly, *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, Pittsburgh, PA.

[10] Seabra Lopes, L. and L.M. Camarinha-Matos (1995) Example Generation and Processing for Inductive Learning in the Assembly Domain, *Proc. 4th European Workshop on Learning Robots*, Karlsruhe, Germany.

[11] Seabra Lopes, L. and L.M. Camarinha-Matos (1996) Learning Failure Recovery Knowledge for Mechanical Assembly, *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems* (to appear).