Interactive Assembly Task Planning and Execution Supervision

Luis M. Camarinha-Matos
Helder Jorge Pinheiro-Pita
Luis Seabra Lopes
UNINOVA / Universidade Nova de Lisboa
PORTUGAL

Elsa Rubio
Juan Manuel Ibarra Zannatha
Alejandro Malo Tamayo
Centro de Estudios Avanzados del IPN — Mexico
MEXICO

Abstract: A bilateral cooperation work between UNINOVA/UNL and CINVESTAV in the area of Interactive Task Planning for robotized assembly is presented. The goal is to achieve an interactive planning system that combines automatic plan generation techniques with external interactive help from a human expert. The supervision of execution of the generated plan, including monitoring, diagnosis and error recovery, is also considered.

1. Introduction

An assembly task can be represented at multiple levels of abstraction. In general terms, assembly planning/programming is a set of activities that produces a multilevel task description, going from an abstract level of specification (for instance product structure) onto refined lower abstraction (more detailed) levels, ending by an executable task specification.

Planning in Manufacturing and Assembly is typically liable to a hierarchical approach. The lowest level includes detailed planning and execution processes, like shopfloor control or process planning. An intermediate level includes operational planning processes like master production scheduling ,materials requirements planning and capacity planning. At a more abstract level, processes related to long-term planning, such as strategic planning, marketing planning, financial planning, production planning and high-level performance monitoring, will be considered [4].

Several attempts at adapting generic planners, developed by the AI community, to realistic robotic tasks have been made[1][2][3]. Most of the planners were conceived having the "blocks world" in mind and some approaches used experiments in a very particular situation, that could be accommodated to a very simplified world model. Other approaches are strongly geometric-reasoning-based, requiring heavy processing procedures. On the other side, the most adequate spatial-related solutions are not completely justified by pure geometrical reasoning but depend on other technological constraints. Work has still to be done in combining geometrical reasoning with product design, technological and process-planning knowledge (concurrent engineering approach).

It is also important to note that the required knowledge is available with different human experts [designers, process planners, cell programmers, etc.]. In that sense, an interactive approach, where different human experts interact with a knowledge based planning system, seems to be the natural approach. The objective is to assign the system the tasks which an expert finds more difficult and the human the tasks which he solves more efficiently — anthropocentric approach. On the other side, it is worth noting that any planning strategy is always based on simplified models of the world. When dealing with physical systems, one has to cope with exceptions or errors not anticipated during the plan generation. Therefore, a planning system has to be complemented by an execution supervision module, capable of real-time decision-making.

Typical industrial solutions for robotized assembly are essentially non- flexible solutions, imposing a high degree of cell structuring in terms of feeders and fixtures and being able to cope only with a reduced number of variations. However, nowadays competition among companies imposes faster adaptation to new market needs, which implies quick adaptation of existing production resources to new products or product variants. Therefore, flexibility has to be achieved by relaxing cell structuring constraints and improving cell control programs in order to adapt to new applications. Interactive plan generation is an approach to reducing re-programming efforts.

However, in less structured environments, it is difficult to foresee all the events that might occur. Since it is desirable that a system works autonomously for as long as possible, control programs must be able to make decisions during execution time according to external asynchronous events and to a given monitoring strategy. In particular, flexible assembly systems will have to cope with execution failures.

In the proposed approach, a hierarchical task decomposition is adopted. In parallel with the hierarchical representation of the task, the architecture includes an execution supervisor. It provides, at different levels of abstraction, functions for dispatching actions,while monitoring their execution, diagnosing and recovering from failures.

2. Planning and Representation

2.1. Interactive Planning

During the last years, the Center for Intelligent Robotics of UNINOVA / UNL has been working on the subject of Interactive Planning. The initial work aimed at applying this concept to motion and assembly operations [3, 4]. The idea was to create a multilevel architecture where an assembly task was to be represented at different levels of detail. Depending on the abstraction level, the system asked for some help from the user to translate to the next level. This help could be simple information using some textual interface or even positioning information [grasping, approaching, trajectory skeletons, etc.] using a graphical interface. In that proposal the graphical simulator has not only been a way to verify / evaluate a generated plan but also an instrument to help the interactive construction of such plan.

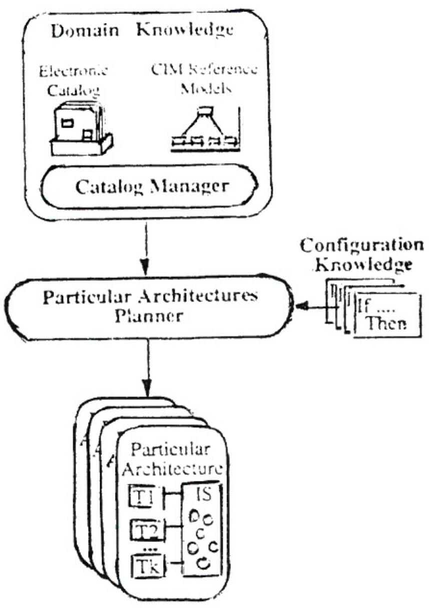


Figure 1 — General Model of CIM-CASE

Later on, deeper investigation was made to extend this concept to the generation of CIM software tools configurations- Particular architecture in terms of CIM-OSA (Open Systems Architecture for CIM) [7][8]. The concept of **CIM-CASE** (Figure 1) was proposed by UNL/UNINOVA team [5][6][10] as an evolution of the toolbox system. The main goal of CIM-CASE is to cooperate with an expert on the derivation of a Particular Architecture (interactive planning). This architecture derives from an Electronic Catalog of CIM tools' models, based on a system-validated user specification. Next paragraphs introduce a very brief explanation of the model of CIM-CASE.

A **Particular Architecture** is a configuration of software modules/applications for a particular CIM system, comprising a set of tools (selected from the catalog) integrated via an Information System (IS). The assumed integrating infrastructure includes, besides the IS and distributed information access functionalities, a **Control or Administrative Architecture**, i.e. an abstract model of the functionalities of a particular CIM system and desired manufacturing processes, used to check the overall behaviour of the integrated system. In close connection with the catalog of tools, does another central component of the system come, a metaknowledge base containing knowledge about CIM activities, application areas, reference information concepts and configuration knowledge — **CIM reference models**.

A **Catalog Manager** module is designed to support the maintenance, graphical browsing and explanation facilities associated with the catalogue and reference models. Such functionalities are even more useful if thinking of the interdisciplinarity involved in CIM models.

A second functional component of the CIM-CASE concept is the **Particular Architectures Planner**, an interactive support system for specification and derivation of particular architectures, including tools selection and derivation of shared IS concepts.

The main objective of this work was to investigate ways of achieving a specialized **CASE** system to help [interactive planning] in the development of CIM systems.

The first prototype was developed by our group in the context of CIM-PLATO, an ESPRIT project of the European Communities, involving 14 partners (universities and companies) from 7 countries. The main objective of CIM-PLATO was the development of an industrial toolbox prototype consisting of computer-based procedures and tools which support the design, planning and installation of FMS and FAS systems in a CIM environment [11].

Returning to the roots of this research, the goal is, now, to adapt the concepts developed in CIM-CASE to the problem of interactive task planning in assembly. An analogy of the two application areas will show a strong correspondence of functionalities for the two planning systems.

Typically, strategic and operational planning levels will produce manufacturing orders specifying the kind of product to make, the manufacturing operations to use, a precedence graph determining the possible ordering of the operations, the size of the job and the order due date. Taking into account the precedence between manufacturing operations and the kinds of resources that will supply them, a detailed plan will be drawn up. Taking this plan as input, and considering the size of the job, the order due date, other jobs that will be running concurrently, the available manufacturing resources and some optimization

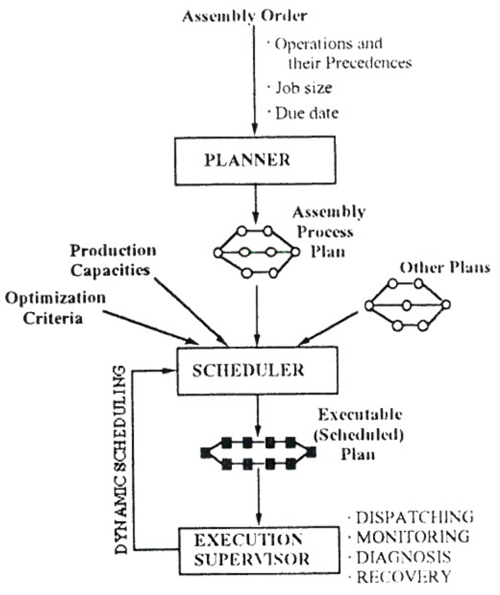


Figure 2 - Framework for Planning and Execution

criteria, a scheduler will produce the executable plan (Figure 2). Execution will be carried out by a Supervisor, i.e. a control program that will receive as input an executable assembly plan generated as described above and will carry out its execution, performing monitoring, diagnosis and recovery functions.

It is important to discuss the structure of the executable plan. Our approach supposes to have hierarchical plans, since, in this way, planning and supervision activities get modular. On the other hand, since a planning activity is often carried out hierarchically, drafting requires no additional effort. From the supervision point of view, the hierarchical approach can be combined with concurrent execution at each level. In this work, fine motion and compliant operations, like peg-into-hole, are considered primitive actions.

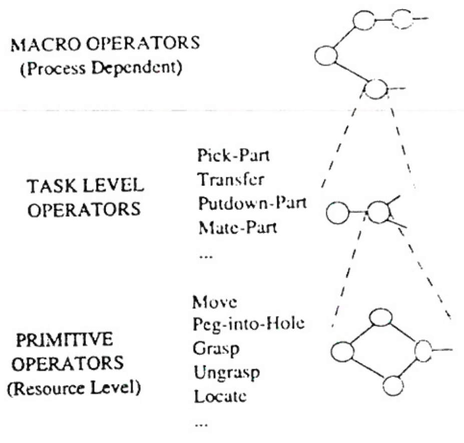


Figure 3 - Hierarchical Assembly Plans

The multilevel structure is illustrated in Figure 3. The internal frame-based representation is illustrated in Figure 4. As mentioned below, the Petri nets based representation will take this structure into account. At the lowest level of the plan, each action is performed by a resource operator. Robot operators, for instance, will perform those operations available at controller level, like Move, Approach, Depart or Peg-into-Hole. The robot may inherit from the currently attached gripper operations as Grasp and Ungrasp. If a robot is equipped with a vision sensor, it will also inherit such operations as Locate-Object. Actions at the next upper level will be performed by task operators which are responsible for performing mate operations: Pick, Putdown, Transfer and Mate.

A variable number of plan levels may manifest in addition to those just described. At each level, the action is performed by the assembly macro operator. These plan levels reflect directly the logical regions of the final assembly and are used to guide the monitoring, diagnosis and recovery steps of the supervision process. In this way, it is assumed that, after the conclusion of some logically important phase of the assembly activity, a monitoring activity should start. On the other hand, a hierarchy reflecting the logical phases of the assembly activity should, in principle, give a more suitable information context to diagnosis and recovery.

The defined categories of operators use STRIPS style (Figure 4). Each operator is an object having: the name of a method which implements the desired functionality; a relation to the resource(s) able to perform that functionality; and the operator parameters, preconditions, add list and delete list. All information related to operators will be included in an Electronic Catalog as the CIM-CASE catalogue.

Relevant information in a plan node includes: the operator that performs it, parameters, pre-conditions, goals, the next and previous operations and the next and previous plan levels.

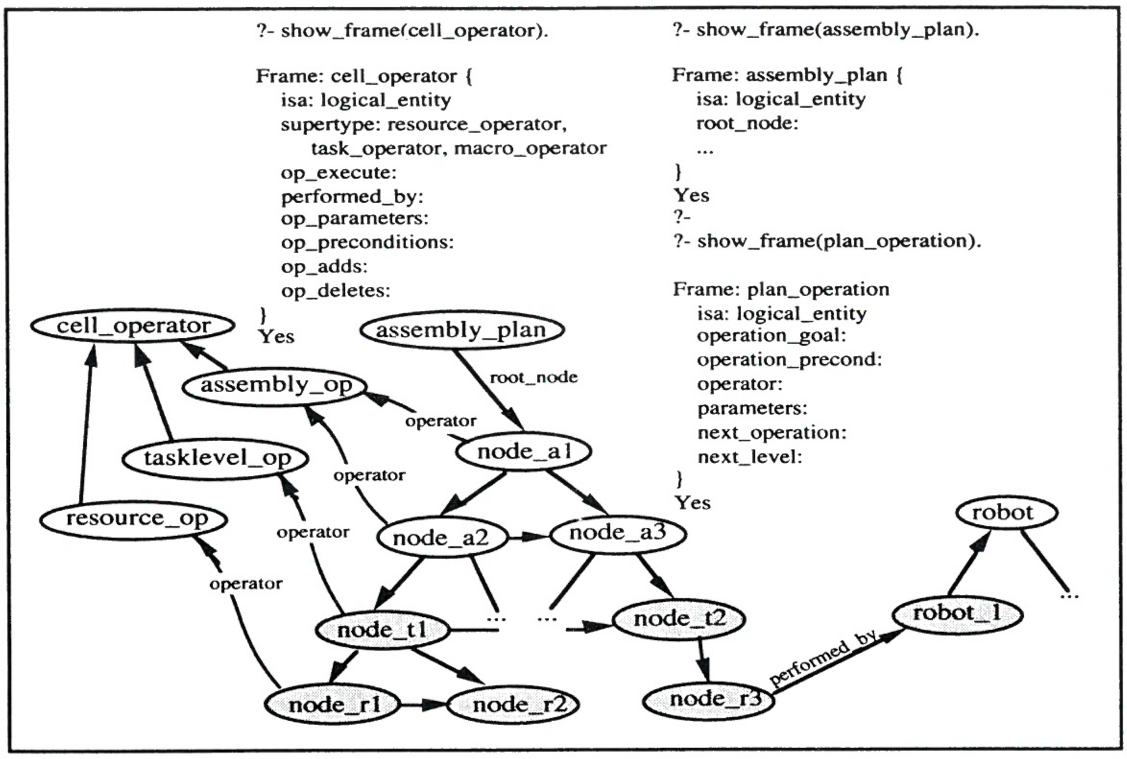
As already mentioned , the generation of a configuration or plan by CIM-CASE is an interactive process that uses a set of reference models. In this application, the reference models are models of process plans, products, assembly resources, sensors, monitoring conditions, etc.

The operators' electronic catalog and the reference models will make the domain knowledge of the assembly planner.

2.2. Petri Nets Based Representation

The ROVISA group of CINVESTAV has considerable experience in using Petri nets to model manufacturing systems. A prototype simulator has been developed by ROVISA and applied to various assembly tasks.

Our approach was to explore the Petri nets formalism to represent the assembly tasks whose



```
?- show_frame(cell_operator).

Frame: cell_operator {
   isa: logical_entity
   supertype: resource_operator,
       task_operator, macro_operator
   op_execute:
   performed_by:
   op_parameters:
   op_preconditions:
   op_adds:
   op_deletes:
}
Yes
```

```
?- show_frame(assembly_plan).

Frame: assembly_plan {
   isa: logical_entity
   root_node:
   ...
}
Yes
?-
?- show_frame(plan_operation).

Frame: plan_operation
   isa: logical_entity
   operation_goal:
   operation_precond:
   operator:
   parameters:
   next_operation:
   next_level:
}
Yes
```

Figure 4 - Internal Representation of Operators and Plan Related Concepts

(hierarchical) structure was described above. Let us consider some illustrative examples.

During the assembly of a product, the operations must follow a given order. Depending on the structure of the product, some operations can be done simultaneously. This implies some kind of parallelism. As the assembly proceeds, subassemblies must be ready in order to continue with the assembly (synchronization problem).

That means that some operations can overlap, while others must be sequential. The assembly process is the sum of various sequential processes. In this context, Petri Nets seem to be an adequate formalism for (externally) representing the assembly plan. This kind of a representation has the advantage of facilitating the assembly process automation.

For example, the two representations of the assembly plan of a motor in Figure 5, have different levels of expressivity. The horizontal axis measures the complexity of an element of the assembly. The complexity of the assembly lies in the number of its components. So, at the lowest level there are the single pieces and at the highest level, the complete motor.

The vertical axis measures the degree of detail in the assembly description. At the top of the drawing there is the lowest descriptive level, and at the bottom, there is a higher descriptive level, since the information delivered by the top drawing is that "the motor is an assembly of pieces". While the bottom drawing is more descriptive, saying that the complete motor means the existence of subassembly s1 and subassembly s2, plus some joining pieces. And then the assembly a1 operation can be done. In fact, the assembly a1 can be very complex. As seen, at an abstract level **assemble** is the basic operation, although there are many other ways of making an assembly, which the process planning expert may decide on.

The difference between a piece and an assembly, in many cases, might not be very clear. For example, the carburettor can be looked upon as a subassembly. However, if we buy carburettors instead of assembling them, then it can be considered as a piece. A subassembly is therefore some kind of a minimal unit which an assembly plan specification is needed for

In Figure 5, all pieces of an assembly are represented by a single place. However, in many cases, in an assembly, several pieces of the same type are used. In this case we can use the weight of the line connecting the place with the transition as the number of pieces needed for the assembly (Figure 6). So, a mark on a

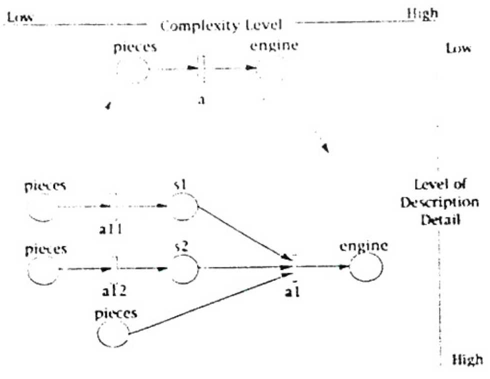


Figure 5 - The Assembly Plan for an Engine

place will mean the existence of a piece. If the mark for a piece does not exist, the assembly cannot be completed. If we want to assemble one engine we need all pieces on the left column. There is also the

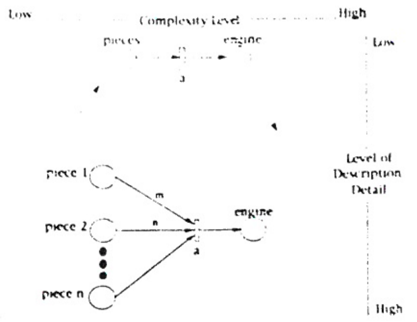


Figure 6 — Different Levels of Description by Type of Pieces

possibility of representing each separate piece as a different place. This might be useful if additional information need be transmitted, for example, the order of tightening screws. However, this comes in a natural way when we refine the description of an assembly operation.

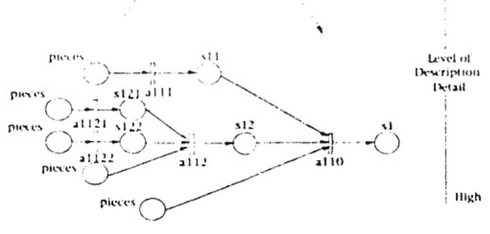


Figure7 - Subassemblies of s1.

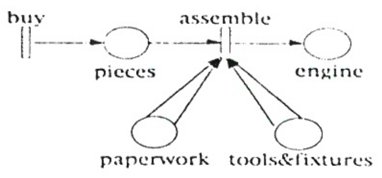


Figure 8 - Assembly Process with Secondary Activities

A more detailed description is given in Figure 7, where the subassemblies of s1 are shown. As the net shows, in order to assemble s1, subassemblies s11 and s12 are needed, as well as other pieces (bolts, washers and nuts). Subassembly s12 is formed of subassemblies s121 and s122. The assembly operation, i.e. a1122, can make reference to a specification sheet, where a Petri net specifies the proper procedure completing an assembly.

Up to now, only the sequence of the assembly operations has been analysed. The way in which the assembly is to be carried out, has not been detailed yet. However, if we really want to assemble an engine, we will also need tools and fixtures.

This is shown in Figure 8. Tooling & fixturing are represented as a secondary flow. So, in this case, Figure says that during the construction process tools&fixtures will be needed for the assembly. Other activities, say paperwork, can be included in a similar way.

This representation of the assembly task gives the order in which it must be executed. Figures indicate this from the left to the right, from pieces to subassemblies, to the finished engine. The abstraction levels used are:

For subassemblies:

- Each subassembly for a subassembly (one place)
- Each type of a subassembly for a subassembly (one place)
- Product (one place)

For pieces:

- all pieces (one place).
- each type of a piece (one place).
- Each type of a piece for one subassembly (one place).
- each piece (one place).

Subassembly A of B means that to assemble B, A is needed. In the same way, piece A of B means that B is a subassembly and A is one of its components.

In case of operations, priorities go from left to right, since the priority is related to the order in which the operations are to be carried out. Their hierarchies go from right to left, being directly dependent on their complexity. Examples of assembly operations can be:

- between two types of pieces,
- between a type of a piece and a type of subassembly,
- between two types of assemblies,

all resulting in a subassembly.

2.3. Interactive Task Specification

As shown in the previous section, Petri Nets formalism seems to fit the hierarchical specification of an assembly plan.

The original CIM-CASE system provides an intelligent editor, based on IDEF0/SADT methodology, for the problem specification [5,7]. Behind this editor some critics are able to validate the user's options, as well as to propose solutions. In addition to a chosen activity, the user has to specify its parameters (Inputs, Controls, Outputs and Mechanisms - ICOM's). Therefore, based on Domain Knowledge , these critics, that belong to the Configuration Knowledge, have the responsibility for maintaining the coherence among different components of the specification.

Pursuant to the philosophy of the CIM-CASE intelligent editor, the approach will be to implement an intelligent Petri Net editor for interactive assembly task planning. Some of the envisaged functionalities are:

Viewing Commands

In any assembly, the starting point is composed of two places with a transition, representing the pieces and the finished product. There is a possibility of expanding and collapsing places or operations. The operation "expand places" shows the user all the pieces as different places (bottom Figure 6). The operation "collapse place" reduces all the pieces to a place (top Figure 6). The operation "expand" expands the assembly operation onto the next level of detail of

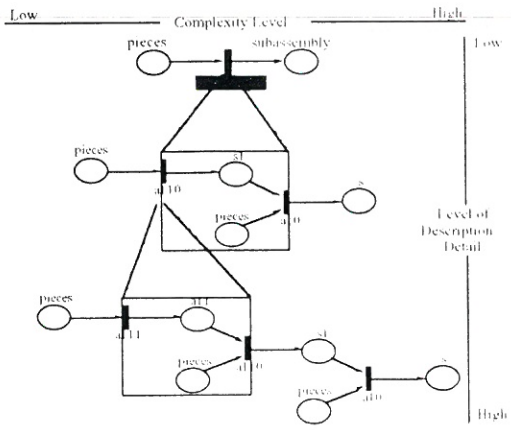


Fig. 9 — Create Operation Applied Once (middle) and Twice (bottom)

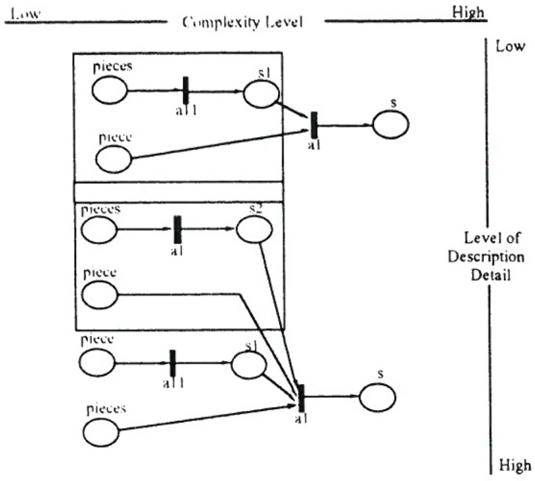


Figure 10 — Parallel Assembly Creation Operation
(above) Selection, (bottom) Result

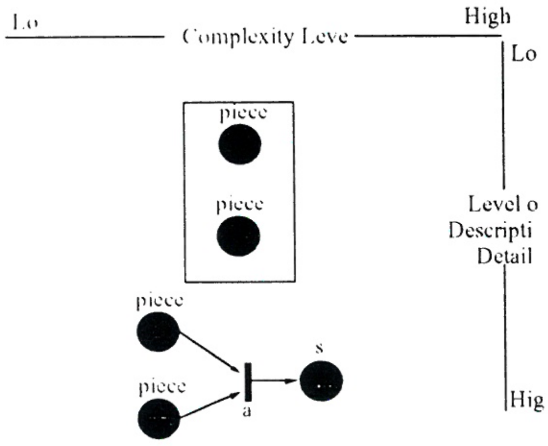


Figure 11 — Assembly Plan for Two Pieces; (above) selection (bottom) result.

description (bottom Figure 5). The operation "collapse" collapses the places and operations (top Figure 5). We can also have "Expand All" to show the network, and "Expand Subassembly", and their opposite commands "Collapse All" and "Collapse Subassembly".

Editing Commands

These are commands that allow us to specify the way an assembly must be completed. For example, in Figure 9 [create operation] two places are inserted and the transition is divided into two. One place represents the pieces joining the assembly, and the other place represents starting assembly, the right one finishing assembly [Parallel Assembly], and an entire new branch (Figure 12).

The latter has been a top-down approach. There is also the inverse possibility, i.e. starting from pieces and working upwards to subassemblies, we would have a finished product in the end. Figures 11, 12 and 13 show this process. In Figure 11, the user begins with two pieces and makes an assembly with them. In Figure 12, the user adds piece 3 to the assembly

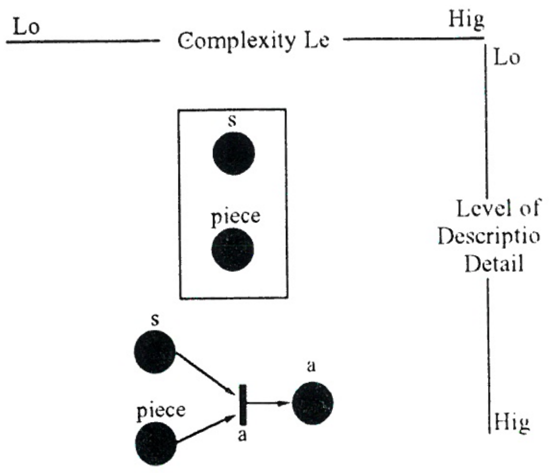


Figure 12 — Assembly Plan for One Subassembly and One Piece; (above) Selection by Window, (bottom) Result.

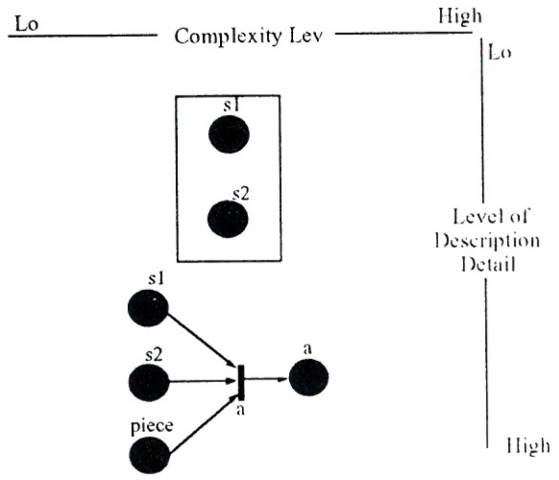


Figure 13 — Assembly Plan for Two Subassemblies by One Piece: (above) Selection by Window, (bottom) Result.

formed in Figure 11. Figure 13 shows the assembly of two subassemblies by means of piece 4 (probably a nut).

These functionalities are adequate for a manual specification and visualisation of the plan. We think that this kind of interface can also be combined with an interactive planning approach, in which assembly domain knowledge can be used to guide the human expert, to propose partial solutions and validate decisions and global coherence (criticism rules).

Criticism Functionalities

These are functionalities that validate different options made by the user during the specification process. Some examples are given below:

- Geometrical restrictions: Based on the product model some rules can test whether one piece may be assembled with other piece or subassembly;
- Pre-Conditions analysis: Based on the model of the operators the editor could detect some impossibilities on the operators' sequence;
- Resources analysis: The editor could test whether an attached resource could perform the specified operation;
- Monitoring actions analysis: During the specification phase some monitoring actions will be indicated. Therefore, based on the operation model, the system can detect some incoherences;
- Specification Completeness: When the user intends to get a good specification he can inquire the system whether something is missing;

The combination of a graphical Petri nets editor with the domain knowledge and with the internal frame-based plan representation is a topic for joint development by CINVESTAV and UNINOVA / UNL.

3. Execution Supervision

Within other ESPRIT project (BLearn II), UNL's group is addressing the topic of execution supervision of assembly tasks. The adopted architecture of the Execution Supervisor reflects the hierarchical structure of the plans. For each plan level, the main functions are [12,14]:

Dispatching: Start the execution of operations, distributing them by the executing agents.and taking care of the synchronization and information exchange aspects. This function can be programmed by hand, using Ada, for instance, or can be automatically synthesized. Automatic approaches to derive a control algorithm from the assembly plan are usually based on a Petri-net model [14,17]. In this approach, the dispatching of actions can be implemented as side effects of firings.

Monitoring: Acquisition of sensorial information in order to detect non-nominal feedback from the system, i.e. deviations between the expected state and the observed state. Two monitoring modes are normally considered: discrete monitoring and continuous monitoring. Discrete monitoring is used to check preconditions before and goal achievement after the execution of actions. Continuous monitoring is used to check sensory conditions during the execution of actions. Monitoring rules are one way, that has been used in previous works [12,14,20], to encode the knowledge necessary for monitoring the assembly process:

```
IF <situation> AND <sensory-condition>
        THEN <actions>
```

Example (Pick monitor):

```
IF agent IS robot
AND desired_goal IS part_held
AND grip_sensor IS off
THEN ASSERT pick_failure
```

Diagnosis: Assessment of the execution and classification and/or explanation of exceptions. When the monitoring function detects a deviation, the

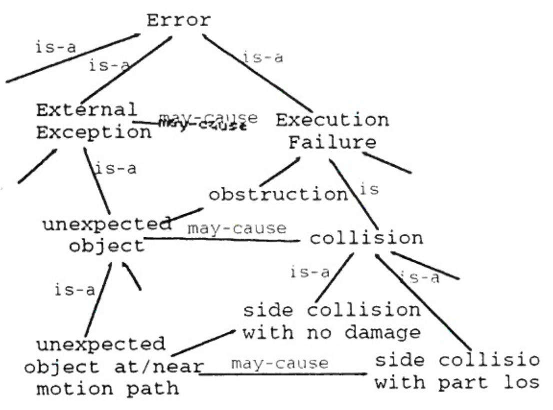


Figure 14 — Example of Causal Links at Different Levels of the Error Taxonomy.

diagnosis function is called. This function will first try to confirm the existence of an execution failure and then update the world model. Then diagnosis will try to classify and explain the failure. Srinivas [18] developed a failure reasoning model for each possible action. This model is a collection of all possible failures in that action and of all features that each failure is expected to manifest.

In UNL three main types of errors have been identified [13,12]: system faults, external exceptions and execution failures. **Execution failures** are deviations of the state of the world from the expected state detected during the execution of actions. For example collision, obstruction, part slippage from the gripper, part missing at some expected location, etc., are execution failures. **External exceptions** are abnormal occurrences in the cell environment that might cause execution failures. For instance, misplaced parts, defective parts and unexpected objects obstructing robot operations might cause all the above mentioned execution failures. **System faults** are abnormal occurrences in the assembly hardware and software resources and in communications.

Each error can be more or less characterized, depending on the available information. The model of errors is, therefore, based on a taxonomy, but will also include cause-effect relations at different levels of abstraction (Figure 5). An inference mechanism to work over this model is still to be created. One of the approaches being considered is the application of machine learning techniques. Two techniques of inductive learning have already been used to generate classification knowledge that can be used to identify the execution failure [12].

Recovery: Attempt at finding an adequate recovery procedure for the diagnosed exception. In most of the approaches, recovery actions are selected from a set of pre-programmed recovery strategies or heuristics [14,19,20,16]. One basic question is how to build recovery strategies. Since the detected error is some unexpected (abnormal) event, the nominal plan is not to be altered. Explanation-Based Learning seems a candidate technique to learn new recovery strategies by generalizing specific examples [21].

This architecture is, to some extent, compatible with [15,16], and with various other proposals found in the literature. When an exception is detected before, during or after an operation by the monitoring function, the diagnosis function will be called to classify and explain that exception. If it is not possible to explain the exception, the diagnosis function of the next upper level will be called. If it is possible to explain the exception, then the recovery function will be called to determine a recovery procedure. If recovery is not possible, the problem is then passed on to the next upper level (Figure 15).

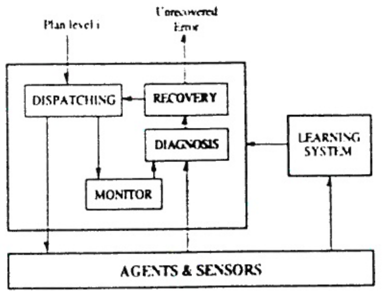


Figure 15 - Execution Supervisor at Level i

Machine Learning techniques are being investigated as an approach to automatically acquiring the monitoring, diagnosis and error recovery knowledge [13].

4. Conclusion

The current joint activity, in the framework of the FlexSys project, aims at combining both skills and results of UNINOVA/UNL and CINVESTAV in order to make a common approach to assembly task planning and execution supervision. The integration of the approaches to model concepts and techniques used by each group seems to be a good way for improving research results of either side.

The workplan includes two main activities. The first one is related to the unification of concepts and techniques used in each group. The development of a common glossary is important for improving communications. The second co-operation aspect is related to the carrying out of concrete cooperation work producing common reports and implementing prototypes to validate the obtained results.

This co-operation work on research is mainly supported by automated exchange of ideas and results, common publications and exchange of people.

The following phases are considered:

(1) Exchange of the most significant papers about the baseline.
(2) Definition of a common reference model and a common knowledge representation.
(3) Evaluation of the possibilities for joint developments and tasks assignment
(4) Co-operative developments, including a demonstrator to present the results of the co-operative activities.
(5) Demos at CINVESTAV and UNINOVA sites.

Acknowledgments

This work has partly been funded by the European Communities (ECLA, PrintSim and Esprit B-LEARN II projects) and JNICT (SARPIC and CIM-CASE projects).

REFERENCES

1. ALMGREN, R., **Planning and Execution of Assembly Operations Based on Blocks World Modifications**, University of Linkoping, LITH-IKP-R-541, 1988.
2. RAMOS, C., **Planeamento e Execução Inteligente de Tarefas em Robótica de Montagem e de Manipulação** , PhD Thesis. Faculdade de Engenharia da Universidade do Porto, 1993.
3. CAMARINHA-MATOS, L.M., **Sistemas de Programação e Controle de Estações Robóticas - Uma aproximação baseada em Conhecimento** PhD Thesis, Universidade Nova de Lisboa, 1989.
4. CAMARINHA-MATOS, L.M. and PINHEIRO-PITA, H.J., **Interactive Planning of Motion and Assembly Operations**, Proc. of IEEE Int. Workshop on Intelligent Motion Control, Istanbul, 1990.
5. CAMARINHA-MATOS L.M.and PINHEIRO-PITA, H.J., **Intelligent CASE for CIM**, Proc. of IEEE Int. Workshop on Emerging Technologies and Factory Automation, Melbourne, 11-14 August 1992.
6. CAMARINHA-MATOS, L.M.and PINHEIRO-PITA, H.J., **Interactive Planning of CIM Software Systems**, Proc. of 2nd UNIDO Workshop on Industrial Robotics and CIM, Belgrade, 4-6 September 1991.
7. CAMARINHA-MATOS, L.M.and PINHEIRO-PITA, H.J., **Interactive Planning in CIM-CASE**, Proc. 1993 IEEE International Conference on Robotics and Automation, Atlanta,GA, USA, 1993.
8. PINHEIRO-PITA, H.J.and CAMARINHA-MATOS, L.M., **Comportamentos de Objectos Activos na Interface Gráfica do Sistema CIM-CASE**, Proc. of 4^{a}s Jornadas Nacionais de projecto, planeamento e produção assistidos por computador, Lisboa, 1993.
9. **Open System Architecture for CIM**, in AMICE, Esprit Consortium (Ed.) SPRINGER-VERLAG, 1989.
10. CAMARINHA-MATOS, L.M., **Toolbox Manager: Towards a Concept of CIM-CASE**, Proc. of the CIM-PLATO Workshop on CIM Planning Tools, University of Karlsruhe, Germany, 25-26 February 1992,
11. BERNHARD, R., **CIM Systems Planning Toolbox - Project Survey and Demonstration**, Proc. CIM-PLATO Workshop on CIM Planning Tools, University of Karlsruhe, 25-26 February 1992.
12. CAMARINHA-MATOS, L.M., SEABRA LOPES, L. and BARATA, J. **Execution Monitoring in Assembly with Learning Capabilities**, 1994 IEEE Int. Conf. on Robotics and Automation, San Diego, CA, May, 1994.
13. SEABRA LOPES, L. and CAMARINHA-MATOS, L.M., **Learning in Assembly Task Execution**, Proc. of The Workshop on Learning Robots, Turin, Italy, 1993.
14. CAMARINHA-MATOS, L.M. and OSÓRIO, L., **Monitoring and Error Recovery in Assembly Tasks**, 23rd ISATA, Vienna, Austria, 1990.
15. CAMARINHA-MATOS, L.M., NEGRETTO, U., MEIJER, G.R., MOURA-PIRES, J. and RABELO, R., **Information Integration for Assembly Cell Programming and Monitoring in CIM**, 21st ISATA, Wiesbaden, Germany, 1989.
16. MEIJER, G.R., **Autonomous Shopfloor Systems / A Study into Exception Handling for Robot Control**, PhD Thesis, University of Amsterdam, The Netherlands,1991.
17. ZHON, M. and DICESARE, F., **Petri Net Sythesis for Discrete Event Control of Manufacturing Systems**, Kluwer Academic Publishers, 1993.
18. SRINIVAS, S., **Error Recovery in Robot Systems**, PhD Thesis, California Institute of Technology, 1977.
19. GINI, M. and GINI, G., **Towards Automatic Error Recovery in Robot Programs**, The International Joint Conference on Artificial Intelligence, Karlsruhe, Germany, 1983, pp. 821-823.
20. LOPEZ-MELLADO, E. and ALAMI, R., **A Failure Recovery Scheme for Assembly Workcells**, 1990 IEEE Int. Conf. on Robotics and Automation, Cincinnati, Ohio, 1990.
21. ZHENG, Y. and DANESHMEND, L.K., **Learning Error Recovery Strategies in Telerobotic Systems**, the 1991 IEEE International Conference on Robotics and Automation, Sacramento, CA, 1991, pp. 252-259.