Control of industrial robots through high-level task programming

Jean-Paul Meynard

INSTITUTE OF TECHNOLOGY
LINKÖPINGS UNIVERSITET

ISBN 91-7219-701-3 ISSN 0280-7971
PRINTED IN LINKÖPING, SWEDEN
BY LINKÖPING UNIVERSITY

Control of industrial robots through high-level task programming

by

Jean Paul Meynard

May 2000
ISBN 91-7219-701-3
Linköping Studies in Science and Technology
Thesis No. 820
ISSN 0280-7971
LiU-Tek-Lic-2000:16

ABSTRACT

In this thesis we present an experimental research platform in robotics, XPROB. This platform has been designed to be a tool that facilitates the development of robotic applications. XPROB achieves a flexible prototyping system that features a task-level programming environment, a dynamic representation of the work-cell's equipment, and sensor data integration at runtime allowing on-line program monitoring and adaptation.

This thesis describes how the object-orientation paradigm combined with a traditional layered-control structure lead to an open and dynamic architecture. It also presents an advanced object representation to handle high-level reasoning, even about partially recognized objects.

The platform was first evaluated using simple robotic applications, such as assembly and sensor-guided actions. Afterwards, an industrial application, consisting of a disassembly line for worn-out electric motors, was successfully set up and controlled by our platform.

This work has been supported by the Swedish National Board for Industrial and Technical Development (NUTEK).

Department of Computer and Information Science
Linköpings universitet
SE-581 83 Linköping, Sweden

To Maria

Abstract

In this thesis we present an experimental research platform in robotics, XPROB. This platform has been designed to be a tool that facilitates the development of robotic applications. XPROB achieves a flexible prototyping system that features a task-level programming environment, a dynamic representation of the work-cell's equipment, and sensor data integration at runtime allowing on-line program monitoring and adaptation.

This thesis describes how the object-orientation paradigm combined with a traditional layered-control structure lead to an open and dynamic architecture. It also presents an advanced object representation to handle high-level reasoning, even about partially recognized objects.

The platform was first evaluated using simple robotic applications, such as assembly and sensor-guided actions. Afterwards, an industrial application, consisting of a disassembly line for worn-out electric motors, was successfully set up and controlled by our platform.

Preface

The work presented in this thesis was conducted at the Department of Computer and Information Science at Linköping University. It is part of the project 'Sensors and Control of Autonomous Assembly', funded by NUTEK[1] within the research program 'Mobile Autonomous Systems'. The motivation for the project was to provide a flexible software platform for the control and development of robotic applications. The research objectives encompass:

- dynamic object-oriented modeling of the work cell to reflect the current hardware status
- a high-level programming environment to ease the prototyping of manufacturing processes
- reactive control system to handle uncertainties, real world changes, and enable program correction at runtime.

Among the issues raised by the project, we focus on the design of an architecture for a robotic prototyping environment. In particular, we investigate the representation, reasoning, and symbolic programming of 3-D objects. This appears essential to allow the end-user to specify robotic

1. Swedish National Board for Industrial and Technical Development.

tasks using abstract representations of the equipment. Another important aspect tackled in this thesis is sensor integration. Sensor feedback has proven to be extremely valuable to increase the flexibility of robotic applications. However, modeling and interpretation of sensor data can still be enhanced.

The results of this thesis are illustrated by a fully operating platform set up at the Assembly Technology Division, Department of Mechanical Engineering, Linköping University.

Acknowledgements

First, I would like to thank Dr. Anders Törne for offering me the possibility of working on this exciting project and giving me great freedom to conduct my research. A special thanks should go to Peter Loborg for his assistance and for our long, vivid, and fruitful discussions. I would also like to thank former and present members at RTSLAB for providing a stimulating research environment.

This work could not have been possible without the collaboration of Björn Karlsson, Gert Johansson and Jan Erik Andersson, who provided me an extremely valuable knowledge in vision and robotics as well as the necessary equipment I needed to achieve this project. Finally, I would like to express my gratitude to Nutek, The Swedish National Board for Research, for financially supporting this project.

Jean Paul Meynard

Contents

Chapter 1
Introduction

The development of industrial robotic systems still remains a difficult, costly, and highly time-consuming operation. Commercial robot programming environments are typically closed systems. They notably support very limited connectivity with other vendor products and are often customized for particular applications [Dac92].

If we consider material handling applications and assembly systems, additional issues emerge. They are indeed tailored for manipulation of identical parts, hence they require substantial modification for different products. Increasing the flexibility raises a great deal of issues in terms of programming, sensor interaction, and object representations.

The design of the architecture presented in this thesis is centered around those issues. Our goal is to have at our disposal a flexible platform for the control and development of complex robotic applications. Our investigations for this project are thus concentrated on high-level programming environments for rapid prototyping and flexible manufacturing. The concepts elaborated in our approach have been implemented and tested through the development of XPROB −an eXperimental Platform in ROBotics.

In the next section, we give a brief account of the contribution of our work. Then we present an introductory overview of the XPROB architecture. Thereafter we describe the structure of the thesis.

1.1 Motivation and contribution

The re-programming of industrial robotic systems is still a difficult, costly, and time consuming operation. In order to increase flexibility, a common approach is to consider the work-cell programming at a high level of abstraction, which enables a description of the sequence of actions at a task-level. A task-level programming environment provides mechanisms to automatically convert high-level task specification into low level code. Active research has focused on task-level programming, but as reported in [Dac92], limitations still exist, notably due to the following:

- No satisfactory translation mechanisms to translate the task specification into low level code exist for the general case.
- The sensor integration remains a problematic, for example, detecting errors during operation, or handling variation in parts.
- The complexity of the real-world scenarios, including, for example, task specification, synchronization, or time constraints, are not always easily representable.

The XPROB platform constitutes an attempt to alleviate these three limitations through a flexible programming environment. It provides a machine-independent programming language. It supports task re-planning based on sensor feedback. Finally, it significantly reduces the burden of the programmer since it supports: 1) an object-oriented 3-D modeling of the work-cell, providing symbolic reasoning; 2) pre-defined and user-defined high-level command libraries; 3) temporal constraints at different levels of granularity.

1.2 Overview of XPROB, an experimental platform in robotics

To get a better understanding of this thesis, we outline in this section the XPROB architecture and its main characteristics.

To increase flexibility in robotic application programming, an eXperimental Platform in Robotics (XPROB) has been developed. XPROB is an open architecture, see *Figure 1.1*, consisting of four key components: a task level programming environment, a robot program synthesis, a real-time command execution module, and a world model.

The task level programming environment provides the end-user with a robot-independent high-level programming language. Specific functions can be added at any time, as the XPROB kernel supports dynamically loadable libraries. A task is specified in a script that is interpreted when the task is run.

Each work-cell component, for example, robot, tool, sensor, or manipulated part, has a 3-dimensional description modeled in an object-oriented database - the world model. The database can handle partially identified objects, and it provides a mechanism to express confidence in data. The values of the objects' attributes are updated throughout the execution of the task-level program.

In the program synthesis module, a task planner translates the high-level commands into low-level robot commands. A motion planner computes the approaching and final gripper position and orientation. Finally, the low-level commands are translated into robot-dependent instructions.

The real-time command execution module forwards these instructions to the corresponding hardware and/or graphically displays them. During the execution of the program, sensor data can be requested. Upon reception, a filter adds a symbolic value to the sensor data. Further reasoning about these symbolic values allows task re-planning at runtime.

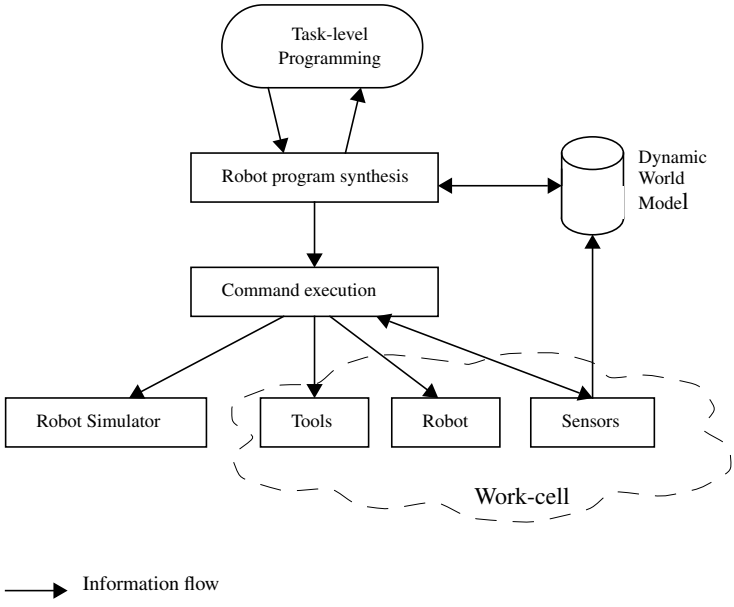


Figure 1.1: XPROB architecture

1.3 Structure of the thesis

The rest of the thesis is organized as follows:

Chapter 2 presents related work in the robot programming domain. We briefly present a classification of the existing robot programming environments. We then describe the basis of the environment which is our focus, namely task-level programming, and comment on some recent approaches.

Chapter 3 introduces the general architecture of our robotic research platform. We detail the platform's key components and the functionality of the prototyping environment.

Chapter 4 first provides an example of assembly of polyhedral objects. An evaluation of the results is discussed. The two following parts illustrate the benefit of sensor integration in robotic applications. Identification of parts by a vision system and service by a force sensor are successively implemented and analysed.

Chapter 5 describes the successful set-up, control and programming of an industrial disassembly line using XPROB.

Chapter 6 presents conclusions and some perspectives for future work.

Chapter 2
Related work

In this part, we briefly present a classification of the robot programming environments. Then we describe the bases of the task-level programming environment. Afterwards, we present some recent contributions to this area. The different approaches are then discussed while focusing on five aspects: programmability, reactivity, maintainability, robustness, and observability.

2.1 A classification of robot programming environments.

As reported in [Pet96], and [Gra89], several classifications have been proposed to categorize the different levels at which robots may be programmed. As task-level programming is our focus, our classification emphasizes the degree of abstraction of robot movement specification.

Robot programming language (RPL) encompasses the joint level and manipulator level. In joint-level programming, the position of the end-effector is specified in terms of joint angles and joint displacement. In manipulator-level programming, the robot motions are specified using pre-defined commands addressing robot-specific area. Today, the manipu-

lator level is still the most widely used programming method employed in industry for manufacturing tasks. The forerunner languages, such as AML [Tay82] or AL [Muj82], have now been superseded by elaborated robot languages like ABB Rapid [Abb94]. Despite their common use, they have two important drawbacks. First, they require specialized knowledge of the language. Secondly, the robot programs have limited portability. As a result, significant investment must be made when changing or acquiring a new robot type or simply when upgrading a new controller from the same vendor. As the number of different robot languages now exceeds a hundred [Nan93], some recent attempts to come up with a 'universal' language have been carried out [Fah98], [Lap99], though not yet having yet a noticeable impact on off-the-shelf products.

Off-line programming environments offer graphical simulation platforms, *Figure 2.1*, in which the programming and execution process are shown using models of the real objects [Rem83]. Consequently, the robot programmer has to learn only the simulation language and not any of the robot programming languages. Examples of off-line environments are Igrip[1], CimStation[2], or RobCad[3], (see [Wit95] for a survey of recent off-line programming tools). Other benefits of off-line programming environments include libraries of pre-defined high-level commands for certain types of applications, such as painting or welding, and the possibility to assess the kinematic feasibility of a move, thus enabling the user to plan collision-free paths. The simulation may also be used to determine the cycle time for a sequence of movements. These environments usually provide a set of primitives commonly used by various robot vendors, and produce a sequence of robot manipulator language primitives such as "move" or "open gripper" that are then downloaded in the respective robot controllers. However, the current state-of-the-art off-line systems suffer from two main drawbacks. Firstly, they do not address the issue of sensor-guided robot actions. Secondly, they are limited to a robot motion simulator,

1. Igrip is a trademark of Deneb, Inc.
2. CimStation is a trademark of Adept Technology, Inc.
3. RobCad is a trademark of Technomatic Technologies Limited

which provides no advanced reasoning functionality, nor flexibility in the tasks.

Figure 2.1: Modeling of an ABB IRB2000 using Robcad
Courtesy of R. Maaloof and Tecnomatrix Inc.

Task-level programming environments enable the user to specify the desired goals of the tasks without defining every movement of the robot in detail. It relies instead on a task planner that generates a reliable plan that is expressed in terms of manipulator motions and actions that are necessary to accomplish each task. Task-level programming tools require a great deal of information about the workcell, the robots, the objects, the initial state of the environment and the final goal to reach. As the collection of information can become extremely tedious and time-consuming, some approaches [Nna93] foster the use of CAD/CAM data, which characterizes the objects themselves and the way they can be manipulated.

2.2 A closer look at Task-Level Programming Environment

We describe in this section the overall architecture of a task-level programming system. This architecture relies on three interrelated components: task specification, world model, and robot program synthesis.

Task specification contains information about the objects being manipulated and the robotic environment. During the execution of a task, the task is specified to the task planner as a sequence of actions on the objects in the world model. Therein, several methods have been developed to specify the different model states: In [Nna93], Nnaji presents the spatial relationships method to describe the relative position of the objects. The most interesting aspect of this approach is that it allows a high level of reasoning about the models, leading to advanced automated planning. A simpler method of task specification is through the description of a sequence of actions. The user can thereby describe the requested tasks directly instead of building a model of an object at a desired position. Another method, proposed in [Lob95,] relies on finite state machines and rules to capture the task specification.

World model contains a representation of all the objects in the workcell (for example, robots, tools, etc.) and their features (for example, the geometric description, the physical description (mass, inertia), or the kinematic description of manipulator characteristics). It must also maintain the position of all objects at any time. Furthermore, the uncertainty for each position and the possible actions on the objects should also be modeled. The data acquisition is clearly a time-consuming operation at the beginning of the application programming, unless this information is made available as part of the design process. In any case, the portability and reusability of each model remain of importance.

Robot program synthesis consists of three major steps: sequence planning, motion planning, and plan checking. Sequence planning typically transforms each task operation into low-level commands. Motion planning is the following stage, which deals with the find-path problem. This problem consists of the determination of kinematically feasible, collision free paths. Despite recent advances [Li95], no solution yet exists for the general case. Finally, plan checking ensures that the intended operations are allowed in the current state of the system, and do not violate any rules.

2.3 Approaches in task-level programming

Several task-level programming-based systems have been developed over the years. This section presents an overview of some recent architectures presenting different perspectives.

2.3.1 RALPH

The RALPH project has been carried out at the Automation and Robotics laboratory, University of Pittsburgh. RALPH stands for 'a Robotic Assembly Language Planning Hierarchy' and is thoroughly described in [Nna93] and [Pri93]. The contributions of RALPH are numerous. In particular, it implements the concept of a CAD-based automatic assembly planner and a hierarchic, robot-independent architecture for generating robot commands from a task specification.

The assembly planner concept enables an automatic plan generation using CAD data extensively. Despite this, it is mainly limited to assembly problems; it is able to plan reliable, collision-free paths for grasp, with fine and gross motions, using geometric reasoning of the symbolic spatial relationships among the objects' features.

The concept of a hierarchic robot-independant architecture introduces a layered command architecture. It breaks down the problem of adapting a high level command to a particular hardware into a set of refinement steps based on the hardware class or type.

Figure 2.2 depicts a simplified representation of the RALPH architecture. We detail in the following paragraph the execution of a task.

The task statement first goes into the parser for grammatical checking. If the task is grammatically correct, it is interpreted and the objects' information is passed to the planners.

The task-level planner first analyses the form of the objects to manipulate searching for their optimal grasping and assembly. Upon completion, it issues a new command with the new data, which is then passed to the

mid-level planner. This interprets the new command and breaks it down into low-level commands (general robot commands) to perform the task. The general robot level planner is robot-type dependent (Scara, Cartesian, etc.) and translates the general robot level commands into generic robot level commands taking into account the particularities of each type of robot. Finally the generic robot level planner translates the robot-level commands into robot-dependent instructions, that is, the language of the controller of a specific robot. Sensor data may be requested and interpreted during the execution of a task.

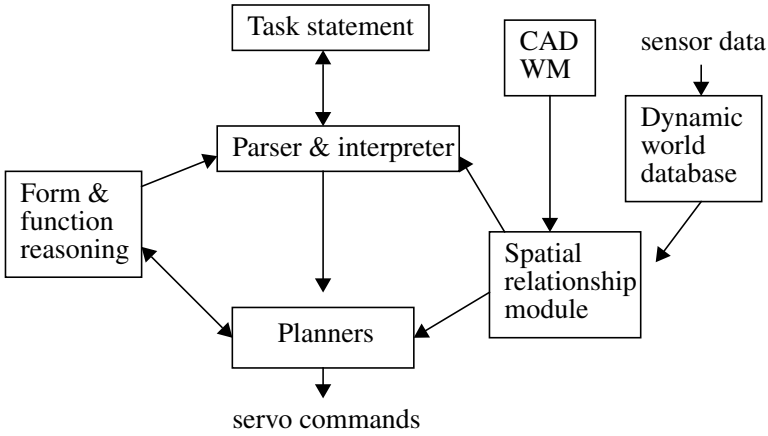


Figure 2.2: Simplified RALPH structure

The main difficulty using RALPH is that it requires extremely accurate models to perform the automatic plan generation. In addition, two aspects are disadvantageous: it gives very little support for error recovery and for the specification of temporal constraints.

2.3.2 STANFORD SMART ROBOTIC WORKCELL

The Stanford Smart Robotic Workcell is a two-arm robotic system developed at the Aerospace Robotics Laboratory, Stanford University. The research around the Stanford Smart Robotic Workcell focuses primarily on motion planning, dual-arm cooperative work, and system design issues. One of the contributions of this project has been to investigate how the various modules of the system, for example, planners, simulators, robots, can be interconnected. In [Par95], the principles of an interface-based design technique are proposed. This approach relies on modular design where a large system is broken into small, well-defined modules with specified functionality. The approach suggested here is that substantial flexibility and faster development time can be achieved if, instead of a traditional subsystem design, where each module is designed and then interfaced to the global system, the interface specification is first defined and the different systems' components (user interface, planner, simulator) are tailored to the software bus and communicate with a bus-access protocol. An overview of the architecture is illustrated in *Figure 2.3*. A database, not represented in this diagram, contains basic information about the objects and the available robot commands. Advanced on-line motion planner strategies for the two-arm workcell are discussed in [Li95].

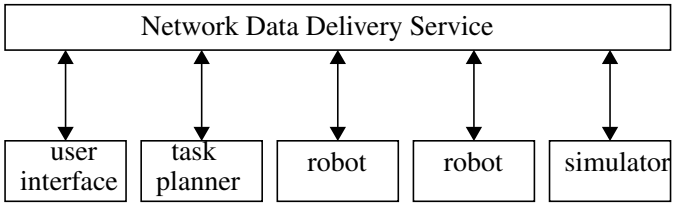


Figure 2.3: Smart Robotic Workcell Architecture

2.3.3 ARAMIS

Aramis is a prototype of a layered design, control, and monitor environment for the overall programming and control of equipment on a shop floor. It has been developed at the department of Computer and Information Science, Linköping University, in collaboration with industrial partners [Lob94].

The Aramis architecture, *Figure 2.4*, consists of three layers, representing different levels of abstraction:

- Task programming level: at the task programming level, the operator specifies what operations should be performed in an abstract model of the physical environment (the world model) and under what conditions, using a graphical hybrid rule-based language. The task specification is then transformed into a Modified Timed Petri Net [Lob95]. The task executes by setting reference values for the objects in the world model.
- Control level: it is made up of computing devices and peripheral hardware connected to the workcell's equipment. Its main task is to keep the real world in a state represented in the World Model, that is, a servomechanism, as the World Model is changed by task program execution.
- Physical level: this is the workcell's equipment.

As an interface between the task-level programming and the control level, the World Model contains an abstract model of the devices on the shop floor. This model is maintained during execution by an active database.

Timing information about real-time algorithms, task frequency, or transition timing can also be added. With this information, it becomes possible to perform a task level cycle-time analysis before runtime, and to schedule activities in order to meet deadlines that otherwise would be missed.

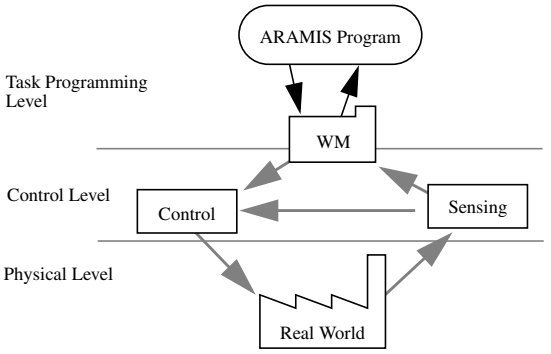


Figure 2.4: Aramis architecture

2.3.4 DISCUSSION

The characteristics of the three previously described systems and XPROB are discussed in this section upon six criteria introduced by Fleury in [Fle94].

Programmability

The user-friendliness of the task specification and world modeling is of major importance as it constitutes the interface between the end-user and the programming environment. RALPH's automatic planning drastically reduces the complexity of the task specification and task calls can be made in textual mode. Special modules are, however, required to acquire and convert the CAD information. Aramis has opted for a graphical interface to enter the rule specification and define the object behaviour. Building an elaborated interface was out of the scope of the present work. However, as we fostered the XPROB interconnectivity to higher control systems, for example, factory control system, we have defined a text-based interface supporting graphical environments, XPROB-View, and script files.

All of the systems presented have been built with object-oriented principles in mind. The object-oriented paradigm satisfies the goals of reusability, portability and extensibility. The review of other environments, such as [Mil91], [Sch91], and [Hwa96] indicates, however, that important variations of the scope of the modeling exist. By and large, they emphasize the object attributes, especially the physical description, leaving aside the object's operations. In XPROB, any entity present in the workcell, and the workcell itself, is represented in a class which embodies static and dynamic attributes as well as parameterized actions that could be performed on the object.

Regarding the independence of the commands from the hardware, RALPH provides a very portable way to translate generic commands into robot / sensor-dependent instructions. In XPROB, we have also adopted this approach and generalize it to any equipment (sensor, robot, tools, devices) present in the work cell.

Reactivity

To cope with the uncertainties and changes in the real-world, external events should be taken into account within given time bounds. Both Aramis and RALPH have stressed the sensor integration in their architecture. As mentioned previously, non-sensor-specific commands are issued in RALPH's task level, thus offering a high level of abstraction to the programmer. Real-time sensor feedback is made possible in Aramis through the specification of time limits on the sensing operations. These functionnalities have been added during the design of XPROB. It also features a symbolic representation of sensor data present in both systems.

Maintainability

One important aspect of the maintainability of a system is its capacity to integrate new functions. Therein, the 'plug-and-play' bus approach presented in the Stanford Smart Workcell is appealing. It is indeed highly relevant to have the possibility to exchange high-level components such as planners to test different strategies and algorithms. Nonetheless, this approach seems more suitable to a distributed environment, whereas the user-interface, planners, and simulator are likely to be on the same compu-

ter. In addition, it is not clear how commercial hardware, usually with limited connectability, can be connected on the bus. In XPROB, we have chosen to adopt a more centralized system, based on dynamically loadable function libraries.

We also introduce the concept of an external supervisory controller. An external controller can be defined as any high-level reasoning or programming system, able to initiate valid commands to the XPROB platform. The underlying idea is to provide a mechanism to enable other systems, offering a higher level of cognition, for example, multi-agent architecture [Nis98], to take over the control of XPROB.

Robustness

The verification of the task specification should indicate not only that the task is feasible, but also that it will be executed within a given time frame. One interesting aspect in Aramis is the possibility to insert real-time constraints in the task specification. After the translation of the task into a MTPN, a scheduling of the execution can be performed. The results then show what execution can be performed in parallel, as well as the minimum and maximal cycle times. Such an advanced functionality is not available in XPROB. However, maximum execution time can be specified for each low-level command and cycle-time computations can be placed at any location in the task specification. Although no formal analysis is performed, simulations give rough approximations. In addition, guards can establish temporal constraints between a sequence of operations or time critical operation. At runtime they will automatically detect missed deadline and take appropriate actions.

Observability

Although the three systems offer simulator connection, they keep a clear separation between the simulation and the real program execution modes. XPROB also offers both modes, but proposes as well a hybrid mode more suitable for prototyping, in which some objects are simulated and others are not. This implementation, discussed in the Prototyping section of Chapter 3, makes the program tuning safer, while keeping the real-time monitoring possible.

Chapter 3
Description of XPROB

This chapter introduces the general architecture of our robotic research platform, whereas the next two chapters present successively basic and advanced applications. XPROB is an acronym for 'an eXperimental Platform in Robotics'. We first introduce the key concepts of the XPROB architecture. We then describe the four main components of its architecture: world modeling, task specification, program synthesis, and command execution. At the end of the chapter, we describe the key functionality of the prototyping environment.

3.1 Outline of the approach

Building an open environment that integrates commercial products and research tools is a challenge in itself. While the former offer very little support for external connectivity, the latter need constant modification and usually suffer from a lack of robustness. This last drawback can become a severe issue when the control of potentially hazardous and costly devices is concerned.

The design of an architecture that accommodates heterogeneous software and hardware leads often to a rather static system. Instead, it should also be possible to dynamically integrate new subsystems (software, hardware) into or remove existing subsystems from the system without stopping and re-initializing the working environment.

The architecture of XPROB is an attempt to address the issues previously discussed. XPROB relies on a reliable core system and a fully upgradable development layer, which allows the end-user to tailor the platform for the application of its concern. To cope with the heterogeneous software and hardware, XPROB implements two key concepts. Firstly, the platform is both application-independent and hardware-independent. Secondly, the object-orientation paradigm is extensively used to model any particular piece of hardware or software as an object. Each object contains detailed interfacing information. Such an approach allows XPROB, for example, to accept, modify, and integrate any hardware definition at runtime. As no device-specific references or instructions are allowed in XPROB, we need to introduce a new programming language. XPROB task-level programming language provides rich and high-level commands to ease the burden of programming multi-vendors' equipment. It offers a syntax close to traditional robot programming languages, but its interpreted execution mode eliminates time consuming compilations. The task interpreter structure is based on the general architecture, and an analysis of previous task-level systems presented in Chapter 2. Though XPROB complies to a large extent with the general task-level framework (world modeling, task specification, and program synthesis), our design introduces two adaptations. Firstly, it fosters a tighter integration of the world model into the program synthesis. Secondly, a real-time control execution module is added to take into account sensor feedback and ensure that the program is executed with respect to the time constraints. *Figure 3.1* depicts the overall platform architecture.

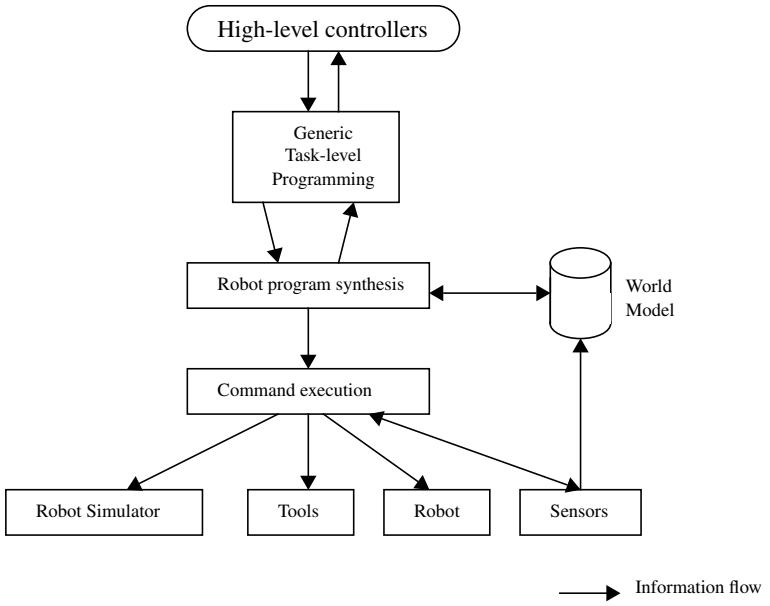


Figure 3.1: XPROB architecture

3.2 World model representation

The object-oriented paradigm applied in robotics has received increasing interest [Mil91], [Mak99], [Fah98]. One of its most interesting assets is that it permits not only tthe storage and maintenance of data, but also the management of procedural knowledge. The advantage of this property is twofold. Firstly, the world model can be updated very efficiently throughout a task program execution. Secondly, we can associate particular program code to objects. In this section, we successively present our object classification, the modeled object information, and how the integrity of the information stored in the database is tackled.

3.2.1 OBJECT CLASSIFICATION

A flexible manufacturing system involves a great deal of equipment that needs to be clearly identified before starting the modeling process. An object classification that efficiently models the physical world, is proposed and discussed in this section. A more thorough study is available in [Bar95] and [Gru94].

Workcell: the workcell embodies all of the equipment: robots, tools, devices, sensors, as well as the manipulated parts. Some equipment may be entered into the world model without having any use for the current application. To prevent any hazardous actions, the authorized hardware should be explicitly specified. Having such information, the task interpreter could then ensure that no reference to incorrect hardware is made.

Parts: a part is any object on which modification or manipulation is performed. Ideally, the workcell should be able to handle different types of parts. To make that possible, each part must possess unique characteristics to allow the system to identify it and apply proper processing on it. Although such an identification may fail due to unexpected conditions, the system should have enough information to decide either to proceed with the current task or request further data about the part.

Robots: several types of robot exist, for example, Scara, which is designed to assemble parts vertically, or Antropomorphic, which has all rotating joints. The kinematic properties of the robot can be represented in a Denavit-Hartenberg matrix.

Tools: a tool is connected to the robot's end-effector. Examples of tools are gluing pistols, grippers or screwdrivers.

Devices: a device is any machine, usually fixed to the ground. They can perform a given action on parts, for example, a press, or assist the robot, for example, a tool exchange system.

Fixtures: a fixture positions and maintains the part so that it remains in the correct position.

Feeders: a feeder supplies parts, carries them from outside the workcell or transports them out of the workcell.

Sensors: a sensor measures input information to operations, or tests the correctness of operations. They can implement various functions ranging from presence detection, identification of objects to quantity measurements.

Controller: a controller is any kind of high level control system able to supervise, monitor or plan the workcell activity.

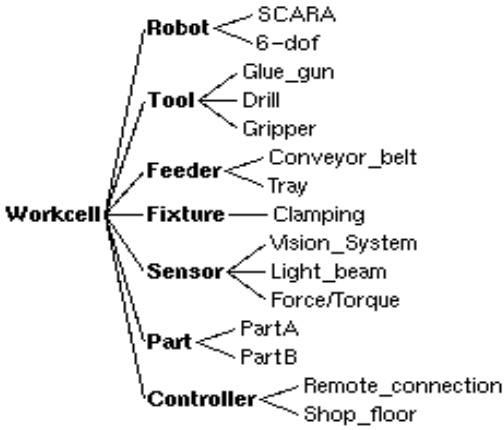


Figure 3.2: Workcell Modeling

3.2.2 OBJECT MODELING

We present in this section the modeling of the objects, that is, what attributes and methods are relevant and how they are modeled.

The objects in the workcell fall into two categories: passive and active. The parts are considered as passive when they can only be manipulated by an active object and they cannot act on other objects. On the other hand, the robot, devices, and similar machines are classified as active.In each class, the attributes and methods are defined with a tuple.

<name, value, state>

Here, **name** is a unique attribute identifier for this class, **value** is the latest value assigned to this identifier, **state** is a mechanism to express confidence in data. In the current implementation of XPROB, we define five possible states, sorted here in a decreasing degree of confidence: set, checked, computed, assumed, unknown. The default state is unknown.

The detailed BNF definitions of the World Model are shown in the appendix.

Table 3.1:

Type of information	**Importance**	**Applicable to**
location	required	all objects
geometric description	required	all objects
feature description	optional	all objects
CAD modeling	optional	all objects
methods	required	all objects
particular characteristics	optional	all object
current status	required	active objects
translation	required	active objects
communication	required	active objects
gripping positions	required	passive objects
attachment	optional	passive objects
sensor-related characteristics	optional	passive objects
filtering methods	required	sensors
kinematic constraints	required	robots

Location: the location is made up of a location identifier and a frame. The location identifier, also called the station, represents the situation of the object in the workcell. The frame is a geometric spatial representation of the object's position. It is made of a coordinate system identifier, the axes of rotation, the rotation values for each axis, and a vector position [Hea86]. This representation is then internally converted into a 4 x 4 homogeneous matrix. This matrix contains a 3 x 3 orientation matrix and a position vector. This popular representation has been preferred to alternative approaches based on quaternions [Fun90] or Euler angles [Cra90]. This representation mode is of great help to express the situation of a point relative to any coordinate system, and to compute spatial transformations by matrix multiplication.

Geometric description: the graphical representation of the object is often subject to discussion and many different approaches can be considered. In the present case, we do not emphasize the spatial relationships among the objects, therefore a detailed geometric object description does not appear necessary, unlike in [Nna93]. We approximate instead the object's shape by a virtual box encapsulating the object. Consequently, three parameters, that is, the length, width and height of the box, are sufficient. The object's center can arbitrarily be set, but the center of the box is, however, usually preferred.

Feature specification: a feature is any physical characteristic of interest on the object. Typically, it consists of a hole, a shaft, or a contact point between two parts. We have adopted the same modeling approach for the feature's shape as for the object's. A frame, having the object's center as coordinate system, defines the position and orientation of the feature.

CAD modeling: each model can be rapidly represented in the graphical simulator by selecting the corresponding model in a library of CAD components. Appropriate parameters must also be added to tailor the CAD drawing. Each 3-dimensional object is represented by a set of lines connecting the object's vertices. The coordinate information is then stored in a vertex table and an edge table. The vertex table contains the coordinate

values for each vertex.The edge table lists the endpoint vertices defining each edge. This scheme is illustrated in *Figure 3.3*. So far the CAD information is used solely to display the objects present in the workcell. The relationship between an object and its graphical representation is in the object specification and formulated as follows:

```
cad {CAD_camera 1.5 1.5 2.5 0.0} set
```

Here, **cad** is the attribute name, **CAD_camera** and the successive values specify respectively the CAD object name and the size parameters, **set** indicates the confidence in the attribute

```
# object 12
points:
1: 1.5 -1.5 0
2: 1.5 -1.5 2.5
3: 1.5 1.5 2.5
4: 1.5 1.5 0
5: -1.5 -1.5 0
6: -1.5 -1.5 2.5
7: -1.5 1.5 2.5
8: -1.5 1.5 0
9: 3 -3 -4.5
10: -3 -3 -4.5
11: -3 3 -4.5
12: 3 3 -4.5
Z axis:
0.0
```

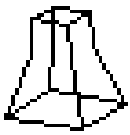


Figure 3.3: Definition and use of CAD models for a video camera

Methods: actions that imply a particular use of the object can be specified as 'methods'. Motion, sensing, or processing instructions are common methods for robots, sensors, and devices, respectively. An action can be anything, ranging from a task-level command to a low-level command through a user-defined function.

Particular characteristics: some attributes are common to only one or several classes of objects and are therefore difficult to classify. These attributes need, however, to be identified within the object.

Current status: any active object can be in any of three states: stop, run, or error. If we consider, for example, this attribute for the "Workcell" object, it indicates whether the workcell is running, stopped, or temporarily disabled. In the last two cases, no command is transmitted to the workcell's equipment until this attribute is set back to a run state. Such a property finds a very interesting use in the strategies set up by the error management module. This allows the execution of the tasks to be temporarily suspended until the recovery mechanism brings the system back to a state in which it can safely be started again.

Translation: any workcell's equipment requires to be programmed in its own programming language. This attribute contains the interface data needed to translate the XPROB's low-level instructions into hardware-specific commands. The stored data details the correct syntax of the command, as well as the optional computation or translation to be performed before generating the final machine code.

Communication: the physical connection of the workcell's equipment to the platform are irrelevant for the task-level programming. This attribute provides a means to define the communication protocol, that is, TCP/IP or serial communication, and a connection identifier, that is, a port number.

Gripping positions: As we have opted for a simplified geometric object representation, the gripping positions of a part cannot be deduced. A grip-

ping position is defined in the object's reference frame by its orientation and location.

Attachment: when two parts are assembled and when a part is disassembled into two pieces, we need to keep track of what has led to the new object configuration. This attachment attribute embodies the necessary data to trace the life cycle of a given part.

Sensor-related characteristics: the use of sensors can drastically improve the flexibility of robotic applications. Common sensor-guided operations are, for example, object identification and location, path planning, or presence detection. In the case of identification, information (color, dimension, weight, or number of features) must be provided in order to either get more accurate sensor data or act as identifying values.

Filtering methods: the data received from the sensor are rarely directly exploitable. The goal of the filtering methods is twofold. The real-world value can first be verified against a range or set of valid values. Assuming that the sensed value is correct, a symbolic value can be associated to it in order to give a higher level of abstraction to the programmer.

Kinematic constraints: the kinematic constraints of a robot are summarized in a table using the Denavit-Hartenberg notation. This provides a compact way to express a robot's joint limits and configuration.

An example of object specification is given in *Figure 3.4*.

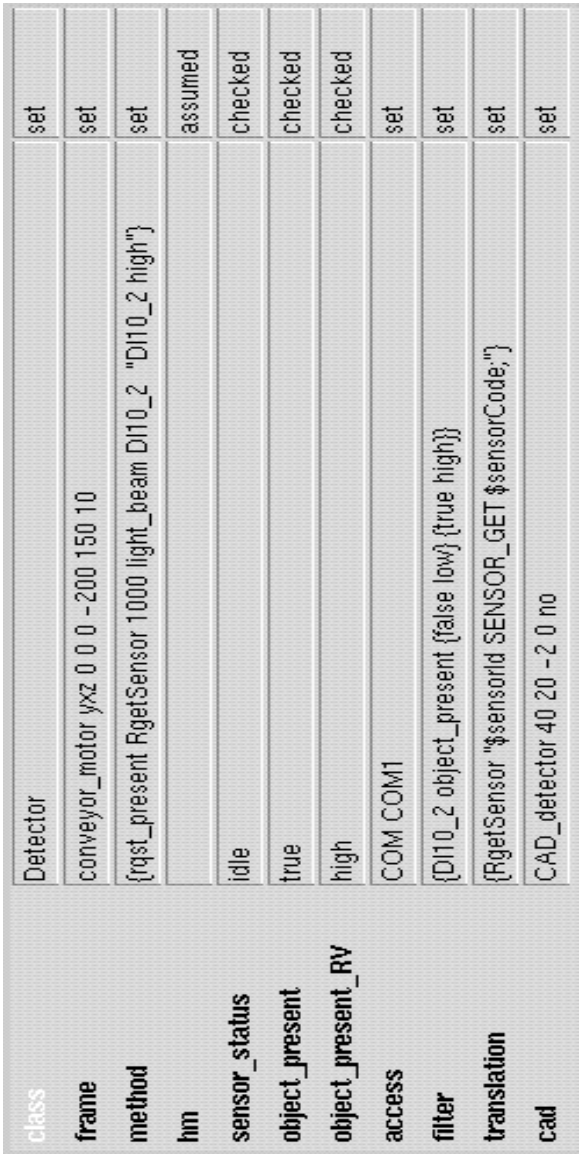


Figure 3.4: Object specification - a light beam sensor

3.2.3 WORLD MODEL INTEGRITY

We describe in this section how consistency and integrity constraints are guaranteed in XPROB.

Consistency Constraints: As the task planner relies heavily on the world model data, it is important that the world model remains in a consistent state. However, programming errors, incorrect object references or incorrect error handling may set some of the workcell components into an unspecified or unexpected state. Consistency rules can be specified at any location in the task-level program. A rule is defined as a triplet <obj, attr, val>, where obj, attr, and val respectively define the object to monitor, the object's attribute to evaluate and the expected value. An example, illustrated in *Figure 3.5*, shows how a consistency rule can be placed in a task specification.

```
proc Foo {robot gripper object fixture} {
   # Consistency check (pre-condition)
   # Verify that the object is currently on 'fixture',the robot has
   # completed any previous move, 'gripper' is attached to 'robot'
   # If it is not the case, the error handler 'err_handler' is executed

Assertion " {$object,station,$fixture}
           {$robot,status,idle}
           {$robot,tool,$gripper}" err_handler
   ...
   # Consistency check (post-condition)
   # Verify that 'robot' has completed the pickup operation,
   # 'gripper' holds 'object'

Assertion " {$robot,status,idle}
```

Figure 3.5: Consistency rules

Integrity Constraints: Defining constraints between the objects of the world model is not yet possible in XPROB. It is left to the end-user to make sure that no illegal action is performed. The simulation environment, synchronization mechanisms and the consistency rules, however, can detect and prevent simple integrity constraint violations, such as releasing a part while the robot is still moving or requesting the vision system to identify an object not yet placed at the right position.

3.2.4 PARTIALLY IDENTIFIED OBJECT

One of the interesting aspects of XPROB is that it allows the definition of an object whose class is temporarily not precisely known. For example, during an identification process, more than one object in the database could match the sensed object. This could happen, for example, if the objects have the same size, if light conditions have changed, or if an object is brought in for the first time. The planner can then have three options: it can stop the task execution, ask for more information, or consider that sufficient information is available at the present time to perform the initial task. While the first solution is not acceptable and should be chosen as the last alternative, an implementation of the two other solutions must be provided. Considering the request of additional sensor data, simple solutions can be obtained through task-level programming. Knowing what are the identifying attributes of an object, the attributes' degree of confidence, and how they can be sensed, an advanced reasoning algorithm can easily be developed. An example that makes use of a vision system is given in Chapter 4.

When detailed information is not required, the previous solution introduces an unnecessary amount of reasoning and actions. Instead we can create a temporary generic object that contains a list of the possible classes. The matching attribute values of the possible classes are assigned to the new object and the attribute status is set to assumed, whereas the non-matching attributes are set to unknown. *Figure 3.6* illustrates the different steps from the creation of a generic object to the refinement process leading to a one-class object.

```
# Create a new generic object of class PART
> OcreateNewObjGen Part coordSystem position orientation location
Part_U_0

# Get the current class of object 'Part_U_0'
> TgetDB Part_U_0 class
Part_A, Part_B, Part_C

# Try to get decrease the number of possible classes with new data
> OrefineObj Part_U_0 dimension "170,170,200"
Part_A_0

# Only the objects of class Part_A have matching dimension.
# A new instance Part_A_0 has been created and
# the object Part_U_0 has been deleted
> TgetDB Part_A_0 class
Part_A
```

Figure 3.6: Partially identified object modeling

3.3 Task specification

Though the task specification is an important phase in task-level programming, designing an advanced task specification interface was out of the scope of our work. The task description is provided in text mode and its syntax complies with the simplified grammar given in *Figure 3.7*.

task_specification ::= **Task** *taskName* **{** *arguments* **} {** *code* **}**

code ::= {*conditional_statement* | *task-level_commands*}*

task-level_commands ::= *TmoveJ_command* | *TsensorActivate* |

TmoveJ_command ::= **TmoveJ** *robotName location*

Figure 3.7: Excerpt of the task specification grammar

XPROB has built-in task-level commands to handle most common robotic operations. The most representative set of commands is the one made available for a robot manipulator. The commands in this package fall into two types of actions: simple and composed actions.

The simple commands are *absolute move* and *object-dependent move.* The effect of these motion instructions is to rotate and translate the robot's end-effector into the desired pose. Subsequently, they will result in two low-level commands: MoveLinear and MoveJoint, as shown in *Figure 3.8*

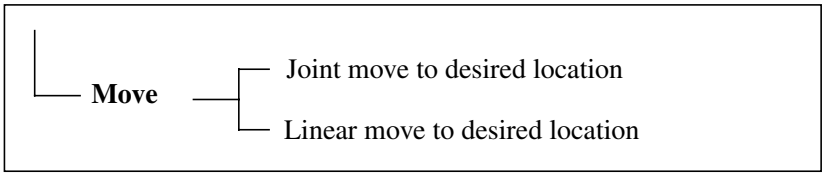


Figure 3.8: Simple robot command

The composed commands are *insert*, *remove*, *pick-up*, and *release.* They comprise the same instruction pattern presented in *Figure 3.9*. Their execution produces a sequence of low level commands of motion and actions such as open/close gripper. The complete description of the built-in task-level and low-level commands is provided in the appendix.

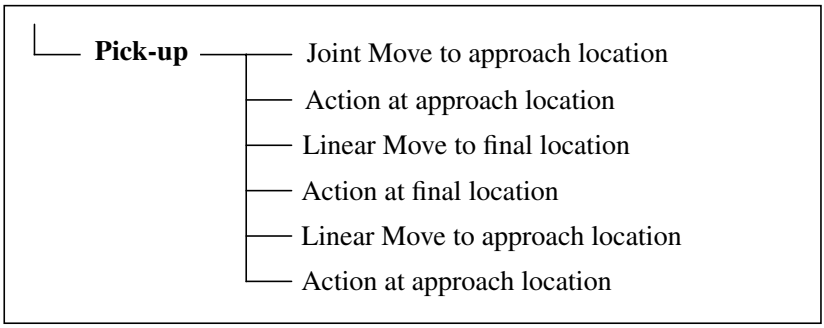


Figure 3.9: Composed robot commands

3.4 Robot Program Synthesis

The robot program synthesis is in charge of breaking down the task specification into a set of low-level commands. It takes a task-level description, which has been entered by the user, and expands it into low-level commands. The final translation into specific robot-dependant instructions is done at a lower level, the command execution module.

This program synthesis module is made up of a task analyser, a task planner, and a motion planner. The module's architecture is illustrated in *Figure 3.10*.

3.4.1 TASK ANALYSER

The task analyser is responsible for syntactically checking the task-level commands (Task Interpreter). It also translates the user-defined task-level commands into generic task-level commands (Task Refinement). The consistency constraints, as described in subsection 3.2.3, are verified at this level.

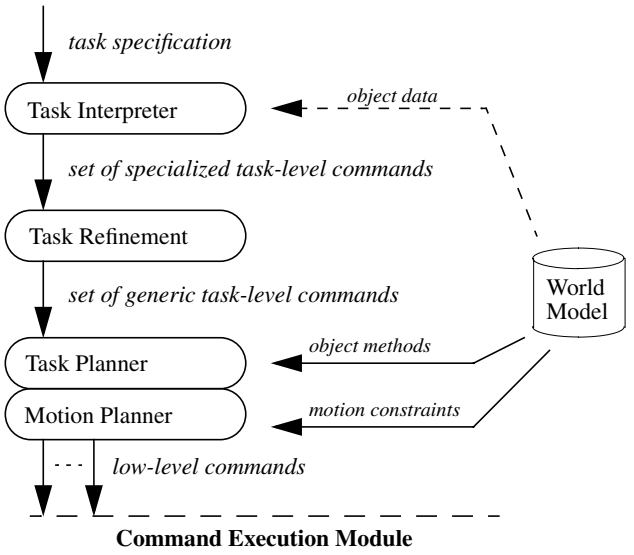


Figure 3.10: Robot Program Synthesis Module

3.4.2 TASK PLANNER

The task planner provides the translation mechanism. It fetches the object methods present in the task-level commands and breaks them down into low-level commands. Prior to this, it performs a series of tests on certain operations, such as assembly or disassembly, to detect any unauthorized action.

3.4.3 MOTION PLANNER

In the literature, the motion planner is often comprised of two parts: grasp motion planning and gross motion planning.

Grasp motion planning deals with the determination of a safe, kinematically feasible, and reachable object grasping. Several approaches have been proposed to implement automatic grasp planning. In the planning

system Handey, noted in [Jon90] and [Loz89], geometric reasoning is applied to polyhedra object. Laugier [Lau90] and Smith [Smi96] have investigated the use of vision sensors to reduce the uncertainty in the grasp when limited information on the part is available. These methods, however do not apply to arbitrary object shapes, which drastically increases the complexity of the planning process.

Gross motion planning computes the intermediate paths. It is also known as the find-path problem illustrated in *Figure 3.11*. For a robot manipulator, it can be defined as the problem of determining of how to move the end-effector of the robot (R) from its current location to fetch or release an object at another location (P), without causing collision with 0_1, 0_2, 0_3. In [Hwa92], Hwang summarizes the general issues and discusses the different research directions in gross motion planning. In the context of industrial robotics, two implementations in the task-level programming environment are discussed in [Li95] and [Nna93]. In both systems, a general configuration space (C-space) is first created. It is then restricted with the joint motion possibilities using the robot kinematic constraints. A numerical potential field method is then applied to find a suitable path. However, no algorithm for the general case exists since all of the state-of-the-art research requires a trade-off between computation time and the generation of short, fast, or smooth paths.

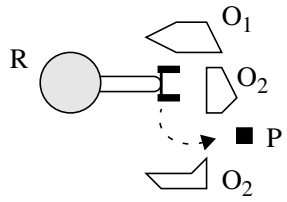


Figure 3.11: The find-path problem

In the XPROB platform, motion planning has been simplified, and improvements are left for later research. In the present state of our system, no automatic grasp planning is available, even if the data necessary to compute the grasping sequence is present in the database. The choice of the grasp and its reachability are the programmer's responsibility. Furthermore, no collision-avoidance algorithm has been implemented. Although these drawbacks constrain the applications' programming, they do not appear critical since the workcell is usually well-defined and the obstacles are well-known. The XPROB's motion planner determines the position and orientation of the end-effector for the equipment approach, grasping approach, and part grasping. These three phases are depicted in *Figure 3.12*. For each piece of equipment in the workcell, we define a safe approach position expressed in the equipment's reference frame. This point is intended to be used by the planner to move from one machine to the other. For a part to grasp, the planner computes the vector normal to the grasping surface. A point is then automatically chosen on this vector at a distance δ from the grasping point, for which the value of δ is specific to the equipment. The computation of the approach vector by the planner can potentially be modified to take into account the equipment's reachability constraints if they are specified.

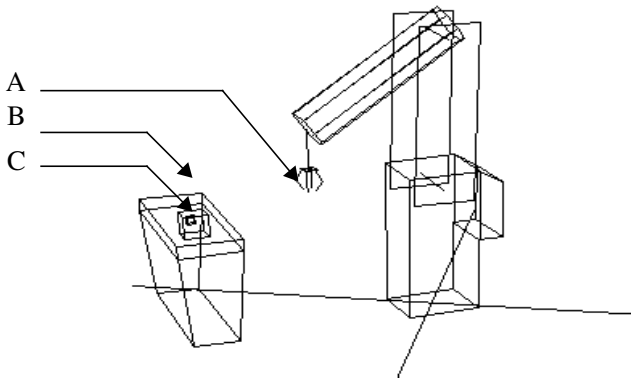


Figure 3.12: Motion planning. (A) equipment's approach location, (B) part's approach location, (C) part's grasp location

3.5 Real-Time Control Execution Module

The control execution module finalizes the task translation by converting the low-level commands generated by the program synthesis module into native machine commands. The translation is not as trivial as it may first appear. This process cannot be limited to a syntax translation. Metric units and angle representation are among the most common differences between robot manipulators. The final command is generated after extracting the command format and its associated code from the database. This code will trigger particular conversions to adapt the initial command parameters to a data format recognized by the hardware.

The CAD data of the objects involved in the command is simultaneously sent to the simulator to reflect the new state of the workcell.

Finally, the execution module encapsulates those commands according to the communication protocol accepted by the hardware. The overall module's architecture is presented in *Figure 3.13*.

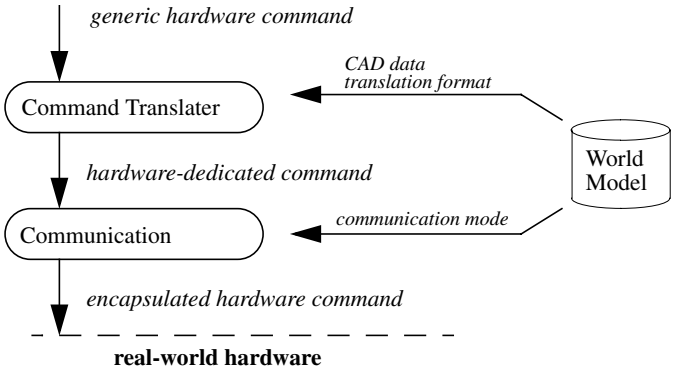


Figure 3.13: Command Execution Module

The execution module is built on a real-time operating system. The maximum execution time associated to each low-level command is used by the execution module to monitor and detect any delayed command acknowledgement. This piece of data is also utilized for various computations by the cycle time manager.

3.6 Sensor Interaction

A simple architecture for sensor integration is proposed in this section and illustrated in *Figure 3.14*. The key idea behind our approach is twofold. Firstly, the programmer should not be burdened by the specific commands addressing the sensors. Secondly, the sensor feedback should also offer a high-level of abstraction in the data representation. If the former issue is solved using task-level programming, see *Figure 3.15*, we need to

make use of the world model to associate a data mapping method to each sensor.

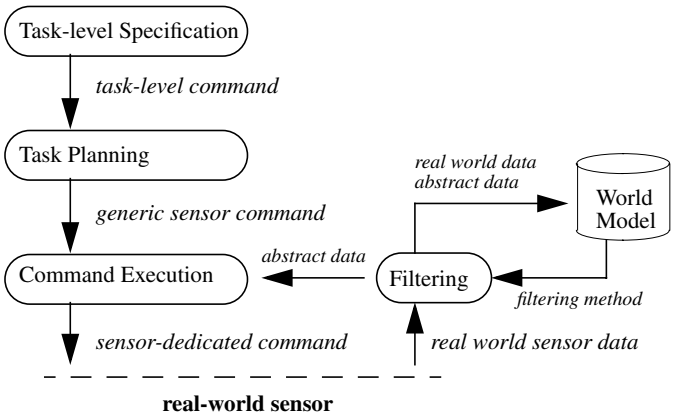


Figure 3.14: Sensor architecture in XPROB

As we consider that several measurements can be performed by one given sensor, we define the attribute 'filtering method' by the tuple: <methodName, sensingCode, associations of abstract and real values>

A filtering method is consequently of the format:

filtering_method ::= *methodName sensingCode* **{** *mapping_value** **}**
mapping_value ::= **{** *abstract-value real-value* **}** |
{ *abstract-value min-value max-value* **}**

The task-level command *TactivateSensor* simply takes as argument the name of the sensor, the sensing method and the name of the attribute receiving the sensor data. This high-level command is then decomposed into a sequence of low-level commands with appropriate parameters, as

indicated in *Figure 3.15*. A consistency check is also automatically performed at the end of the command execution. Still, the sensor could indeed return incorrect data due to unexpected events.

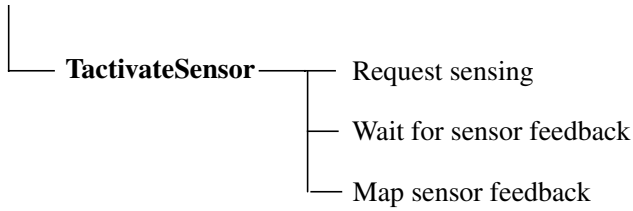


Figure 3.15: Sensing command

3.7 Prototyping environment

We present in this section important aspects of the XPROB platform when used for prototyping.

3.7.1 OFF-LINE AND ON-LINE PROGRAMMING SYSTEM

Several systems make a clear separation between off-line and on-line programming. It is mainly due to the fact that in traditional robot programming, the code is first either generated by CAD software or entered in a text editor. This code is then downloaded on the robot controller prior to execution of it. On-line programming is then used to refine or modify the robot motions. The main advantage of this approach is, of course, that it does not require the use of the workcell's equipment at the development stage. There are also two major drawbacks. Firstly, it implies that the program must be re-generated and re-downloaded for each slight modification of the code. Secondly, the possibilities of task re-planning are considerably limited at runtime. A clear-cut separation between on-line/ off-line no longer exists. In XPROB, we have opted for a hybrid programming environment that combines the benefits of traditional off-line and on-line programming systems. On the first hand, it offers a graphical sim-

ulation of the robot motion in the work-cell. On the other hand, the program instructions are sent at runtime, and the program can be adapted at any time depending on sensor feedback. Furthermore, the objects can be enabled, disabled, or simulated at any location in the program. This means that during the execution of a task, some machines may be simulated, whereas others will be physically activated. This functionality becomes greatly useful when some equipment is not available or must be manipulated with extreme care.

3.7.2 HARDWARE MANAGEMENT

To provide a truly flexible control system, the workcell components should be replaceable and reconfigurable at any time. The programming environment offered by XPROB makes an abstraction of the hardware connection and programming languages of the tool, and then manipulates only references to them. It also provides a simple definition of the hardware topology, letting the XPROB kernel handle the low-level translation and communication functionality. Connecting the workcell's hardware raises several issues. Ideally the communication between XPROB and the equipment could be performed using a high-level communication protocol, such as TCP/IP. This would ensure a fast communication, as well as easy flexibility. However, PLC or detecting devices are unlikely to offer a network interface. In many cases, the simplest and most inexpensive solution is to connect them to robot controllers. Another approach is to connect them to possibly one or several computers via data acquisition cards supporting communication protocols. XPROB provides a simple mechanism to define the physical connection of the components in their 'access' attribute of their world model representation. The example below first depicts a possible workcell layout featuring heterogeneous hardware communication protocol *(fig. 3.16)*. Then, an excerpt of the world model definition of the workcell shows how the physical connection can be modeled *(fig. 3.17)*.

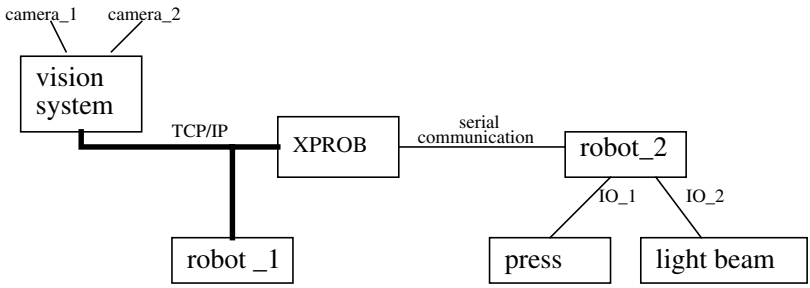


Figure 3.16: XPROB hardware integration

```
SENSOR vision_system
      ...
  // Access mode and portId
      access TCP_IP 3700 set

TOOL press
      ...
  // Access mode and portId
      access COM COM1 set
```

Figure 3.17: Modeling of the harware connection

3.7.3 SIMULATION

To graphically represent the robot motions and the workcell's activity, a simulator, Simderella, has been plugged into XPROB. Simderella is a general purpose public domain robot simulator. It has been developed by Patrick van der Smagt [Sma94]. It consists of three independent programs:

- Connel, which reads the input commands
- Simmel, which is the core of the application implementing the inverse

and forward kinematic algorithms

- Bemmel, which is an X-window based program for robot visualization. It provides a very flexible front-end to display the robot motion

Drastic modifications have been done to the original code to handle dynamic object representation and manipulation, as well as the concept of graspable objects. A library has been written to provide the user the same functionality that the user could have with the initial version. However, XPROB introduces the possibility to write the simulator commands into scripts or insert them at any place in the task description.This graphical simulator, *Figure 3.18*, is based on the wire-frame method, thus high speed can be achieved in the motion sequence. However, it produces an image sometimes difficult to visualize, as no hidden line is removed.

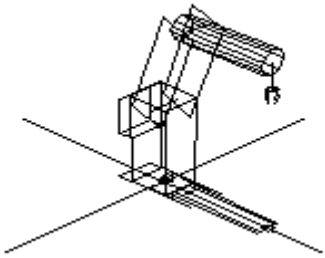


Figure 3.18: Simderella - A graphical 6-dof robot simulator

The simulator is of precious help during development. It can also run concurrently with the robot to compare the behavior of the workcell model with the behavior of the actual system.

3.7.4 ERROR HANDLING

Several failures can occur either due to programming errors, communication or hardware failures. To undertake appropriate corrective actions, the platform must be aware of the current state of each workcell component. We have adopted a hierarchic centralized error management system. Three levels of error exists: warning, recovery, fatal.

- The warning level simply alerts the user that unexpected or suspect values have been specified. For example, floating value instead of integer value as argument in a procedure call.
- The recovery level handles serious failures that can however be corrected. For example, when a failure is detected on a given hardware, preventing it from sending back an acknowledgment, the error handler can disconnect the faulty hardware, issue an acknowledgment to avoid blocking the program execution, and then assign an error flag to the hardware's status attribute in the database.
- The fatal level handles non-recoverable errors. It safely turns off the connected hardware, terminates the real-time tasks, and logs the error before closing the application. Such errors could most likely occur at the initialization stage, that is, when a sequence of tests are performed to ensure the correct set-up of the XPROB real-time kernel.

Chapter 4

Application to basic robotic tasks

To illustrate the concepts elaborated in our work, we present in this chapter three basic robotic applications. In the first application, we introduce the assembly and disassembly of parts and detail the common peg-in-hole operation. In the second and third applications, we present two sensor-based operations. Active vision-based object identification and guarded move are implemented using, respectively, a vision system and a force sensor. In each application, the focus is on the key-ideas it features, therefore the most relevant modeling and implementation aspects are solely discussed.

4.1 Assembly/Disassembly

As reported by Pettinaro, in [Pet96], the assembly of parts is the most common task encountered in manufacturing industry. Two categories of assembly tasks can be singled out: *parts mating,* in which parts are brought in contact with each other, and *part joining,* in which parts are first mated and then joined permanently. In our approach, we mainly focus on the first category. The latter encompasses four sub-types of operation: peg-

in-hole, hole-in-peg, multiple peg-in-hole, and stacking. However, they are extremely similar. The first two types typically represent different variations of the same insertion operation. Multiple peg-in-hole simply adds two new constraints: the simultaneous insertion of all pegs and the line-up of the pegs with their matching holes. Those constraints are, however, not relevant if we consider orientation-dependent peg-in-hole. Finally, at some extent, stacking parts and inserting one part into another are similar actions. A peg-in-hole operation can indeed be seen as putting into contact the bottom surface of the shaft with that of the hole. Similarly, if we assume that a shaft may be flat and a hole has no depth, a stacking operation can come down to a peg-in-hole action. While keeping in mind the assumptions we made, we can design a generic assembly operation that handles the different mating parts. This reasoning also applies to disassembly operations.

4.1.1 ASSEMBLY & DISASSEMBLY MODELING

One important issue in this application is the specification of the assembly. We consider in our approach that this operation consists of putting together one feature of each part. Another assumption is that an object can be assembled using different combinations of objects and features. The result of an assembly is always a third part, automatically added in the world model. Creation-related information is given by two pairs containing the part and the feature identifiers. Assuming that the feature *feat1*of part A is a shaft and the feature *feat1* of part B is a hole, the assembly operation *assembleA&B* has the following format:

```
assembleA&B partA feat1 partB feat1 partC
```

This operation specification contains all of the necessary information to describe a peg-in-hole assembly where feat1 of partA is inserted into *feat1* of partB to create partC. As shown in *Figure 4.1*, the attribute *madeup* of the newly created part PartC-0 contains references to the mated parts.

Inspecting PartC-0		
class	PartC	set
dim	20 80 60	set
station	asm_pt	set
status	unattached	set
frame	world yxz 0 0 0 500.0 100.0 500.0	comp
grippzone	{gripp1 {PartC-0 yxz 0 0 0 0 0 -10 90} assumed}	
grippactive	gripp1	set
feature	{feat1 {PartC-0 yxz 0 0 0 0 10 0 peg 5 5 5} assum	
madeup	PartA-0 feat1 PartB-0 feat1	set
method		assumed
cad	CAD_partC	set

Figure 4.1: World model representation of a part after assembly

A disassembly operation presents the same characteristics, as it takes one part and splits it into two new parts. The disassembleC operation is therefore of the format:

```
disassembleC partC partA feat1 partB feat1
```

4.1.2 HARDWARE DESCRIPTION

The test-bed, depicted in *Figure 4.2*, is made up of a PC, on which XPROB is running over an RT Linux operating system, and an industrial robot, IRB 2400, manufactured by ABB Robotics. An ABB S4 robot controller is connected to XPROB via a serial cable and hosts a Rapid-written program, RAPIDIX, receiving and executing on-the-fly incoming instructions. This basic architecture forms the kernel of XPROB's hardware implementation. This configuration will be reused in the next examples and the disassembly line described in the next chapter.

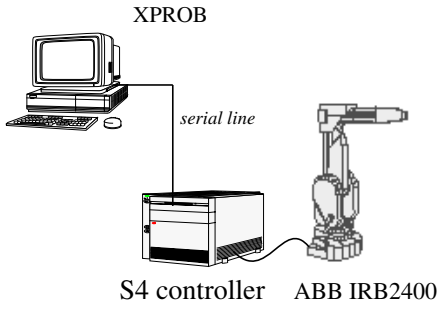


Figure 4.2: Hardware test-bed

4.1.3 DESCRIPTION OF THE TASKS

In the following examples, we assume that the parts are well specified and the operations on them are allowed. Furthermore, to simplify the modeling of the parts, the objects manipulated in this evaluation have a solid shape.

The first task, **Tasm**, described in *Figure 4.3* performs the assembly of two parts. The first two instructions aim to create two parts for the purpose of this example. One of the parts is then grasped at a given gripping position, that is, *gripp1*. Prior to physically inserting the part into the other part, a logical verification and assembly is performed. We must indeed verify that such an operation is allowed for both parts and leads to the same class of part. As the system accepts partially identified objects, it must also be ensured that sufficient information is available. Upon validation, a new part is created in the world model. The next stage is to perform the assembly in the real world using the *TinsertObj* instruction. This instruction takes into account the position and orientation of the target feature to determine the rotation and translation of the gripped part. It can be decomposed in the following sequence of actions. First, the motion planner determines the approach vector, which is a vector normal to the target's feature surface. A point at a given distance from the target's feature is

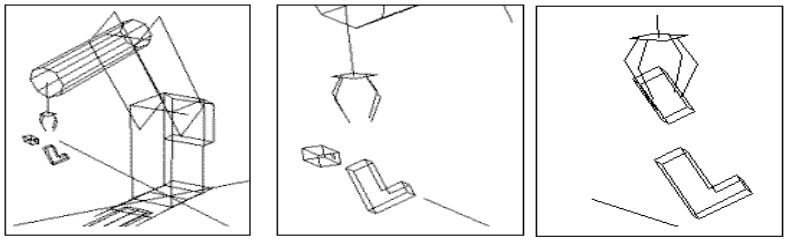
then computed with respect to the approach vector. The rotation to apply on the gripped object is also determined so that both features are lined up. When lined up, the gripped object is moved to the approach position. The next step consists of placing the manipulated part into the second one following a linear path. This guarantees that an optimal assembly is achieved. Once the final position is reached, the manipulator releases the part and returns to the approach position. The example ends with the gripping of the newly created part.

```
Task Tasm {} {
   # Create two parts in the work-cell
   set new_partA [OcreateNewObj PartA world "500 100 500" "0 0 0"
asm_pt]
   set new_partB [OcreateNewObj PartB world "600 150 600" "0 0 0"
asm_pt]
   # Grab one of them
   TpickObj abb_robot $new_partB gripp1
   # Logically assemble them
   if {[Tassemble assemble_feat2 $new_partB $new_partA new_part] ==
"OK"}
   {
      # Insert it into the base in matching their feature #2
      TinsertObj abb_robot $new_partB feat2 $new_partA feat2
      # Pick up the newly created part
      TpickObj abb_robot $new_part gripp1
      return OK
   }
   return error
}
```

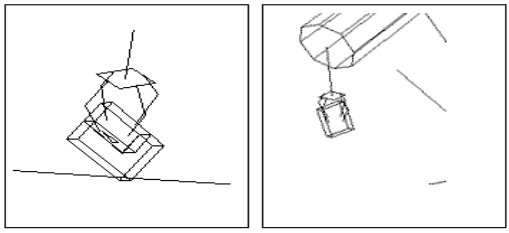
Figure 4.3: Description of the assembly task

4.1.4 TASK EXECUTION

The series of figures, presented in *Figure 4.4*, stems from snapshots from the XPROB robot simulator and illustrates the different phases.



(a) (b) (c)



(d) (e)

Figure 4.4: Assembly sequence. (a) Overall view, (b) Closer view, (c) Approaching phase, (d) Assembly, (e) Manipulation of the new object

The code presented in *Figure 4.5* is the translation of the task-level instructions into robot-specific instructions, in this case ABB Rapid Language.

```
MoveJ [[500.0,150.0,600.0],[0.499,-0.501,0.499,-0.5],
       [0,1,2,0],[0,0,0,0,0,0]],v200,z50;
Set DO10_1;
WaitTime 1;
MoveL [[600.0,150.0,600.0],[0.499,-0.501,0.499,-0.5],
       [0,-1,2,0],[0,0,0,0,0,0]],v200,fine;
Reset DO10_1;
WaitTime 1;
MoveL [[500.0,150.0,600.0],[0.499,-0.501,0.499,-0.5],
       [0,-1,2,0],[0,0,0,0,0,0]],v200,fine;
MoveJ [[500.0,210.0,515.0],[0.0,0.696,-0.697,-0.001],
       [0,-1,2,0],[0,0,0,0,0,0]],v200,fine;
MoveL [[500.0,110.0,515.0],[0.0,0.696,-0.697,-0.001],
       [0,-1,2,0],[0,0,0,0,0,0]],v200,fine;
Set DO10_1;
WaitTime 1;
MoveL [[500.0,210.0,515.0],[0.0,0.696,-0.697,-0.001],
       [0,-1,2,0],[0,0,0,0,0,0]],v200,fine;
MoveJ [[500.0,100.0,590.0],[0.0,0.696,-0.697,-0.001],
       [0,-1,2,0],[0,0,0,0,0,0]],v200,z50;
Set DO10_1;
WaitTime 1;
MoveL [[500.0,100.0,490.0],[0.0,0.696,-0.697,-0.001],
       [0,-1,2,0],[0,0,0,0,0,0]],v200,fine;
Reset DO10_1;
WaitTime 1;
MoveL [[500.0,100.0,590.0],[0.0,0.696,-0.697,-0.001],
       [0,-1,2,0],[0,0,0,0,0,0]],v200,fine;
```

Figure 4.5: Generated robot-specific instructions

4.1.5 EXTENSION TO DISASSEMBLY

A disassembly operation, Tdisasm, presented in *Figure 4.6*, realizes the opposite action. The part is first logically disassembled into two new subparts with respect to the chosen disassembly method. The position and orientation of the new parts are then deduced from the initial part. Once the part models are updated, the manipulator has enough information to grasp one of the newly created parts (*TremoveObj* function). The approach position is also a function of the feature pose configuration and is similarly computed. At the end of this task, the part that has been removed is simply released.

```
Task Tdisasm {} {
   # Create a new part in the work-cell
   set new_partC [OcreateNewObj PartC world "500 300 500" "0 0 0"
asm_pt]
   # Logically disassemble the part
   if {[Tdisassemble disassembly_feat1 $new_partC new_partB new_partA]
         =="OK"} {
      # Grab one of the part out of the initial part
      TremoveObj abb_robot $new_partB gripp1 feat1 $new_partA feat1
      # and then release it
      TreleaseObj abb_robot $new_partB asm_pt
      return OK
   }
   return error
}
```

Figure 4.6: Specification of the disassembly task

A few snapshots from the robot simulator, *Figure 4.7*, depict the successive disassembly phases.

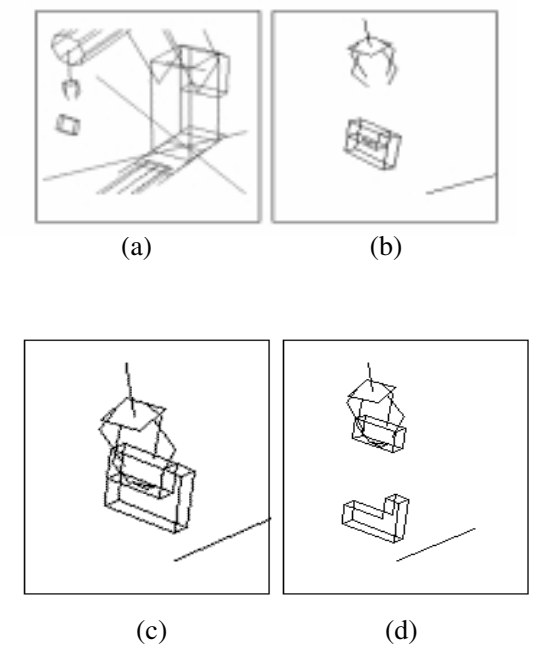


Figure 4.7: Disassembly sequence. (a) Overall view, (b) Approaching phase, (c) Disassembly in progress, (d) One of the parts is removed.

4.1.6 DISCUSSION

The assembly and disassembly operations constitute the most frequent robotic tasks. The two examples presented in this section bring out the modeling, specification, and execution of those basic operations until the final generation of robot instructions. It also shows the limits of the current implementation of the task-level languages. No automatic grasping planning is available.

4.2 Vision-aided robot programming

Today, the use of vision systems in automatic assembly is still very limited. The main applications remain part detection, quality control, pose estimation, and to some extent part recognition. A great deal of academic research has been done in active vision, 3-D vision techniques [FAU93], [SAN97], [RAH96], vision-guided grasping [Smi96], and corrective algorithms [POP94]. However, the off-the-shelf robotic vision systems, for example, Optimaster[1], are still based on 2-D CCD camera, offer limited programmability and are strongly hardware-dependent. As reported in [Kie93], many problems limit the potential gain of using vision systems. The contrast between objects and background, the illumination and reflection problem, can drastically decrease the performance. In addition, all of the systems are color sensitive. This implies that a filter successfully used on one type of object may give very imperfect results when applied on another type. When considering a feature identification process, the vision system should have knowledge of a great deal of object properties, such as the estimated feature position, dimension, or color, to give optimal performance. This supports the idea that a system offering a closer interaction between the vision system and the robot program must be proposed.

In the current implementation of XPROB, the vision-based applications can only use open-loop control. This, of course, limits the range of the application domains, but it facilitates the utilization of commercial programming tools to build complex vision systems.

4.2.1 DESCRIPTION OF THE TASKS

In this example, our focus is on object identification. In the context of flexible manufacturing, parts of a different nature are conveyed in the work-cell. They are then identified and handled. A problem arises when the identification does not produce a single type of part, but a list of potentially matching parts. Instead of requiring an operator to find out the proper object classification, we propose an automatic object identification,

1. Optimaster is a trademark of Sensor Control AB

which combines a vision system, a robot manipulator, and our robotic platform.

The key-idea is to place a part's feature to be identified, for example, a shaft or a hole, right under the camera so that the vision system then analyses a snapshot of the feature. If the feature does not exist, a refinement of the type of the potential part produces a new list of parts having such a feature. The process iterates until only one single type is possible or when all of the features have been tested. The implementation of the task is not trivial and encompasses a great deal of reasoning about the manipulated object.

The task, Tidentify, presented in *Figure 4.8*, implements the object identification. We first declare a new unknown part and refine it with no criteria. The result of these operations is a part having all of the possible part types of the world model. Our application requires that the part can be grasped by the robot manipulator. For this reason, the next step is to check if such a position is defined. If this is the case the object is gripped and the identification process starts.

```
Task Tidentify {} {
    set new_part [OcreateNewObjGen Part
                  conveyor "-200 600 50" "0 0 0" asm_pt]
    OrefineObj $new_part ""
    if {[TassertionS "{$new_part grippzone.gripp1 unknown}"]} {
        puts "Grasping position unknown"
    } else {
        TpickObj abb_robot $new_part gripp1
        TtestFeat  $new_part abb_robot
    }
}

proc TplaceObj {robot obj obj_feat base base_feat} {
     OinsertPart $robot $obj $obj_feat $base $base_feat
        ""          # no action1
        "TactivateSensor vision_lab rqst_feat_present feat_present"
        ""          # no action3
        -100        # approach distance
}
```

Figure 4.8: Description of the identification task

The instruction TtestFeat is another specialized task-level command that recursively requires the vision sensor to assess the presence of each object's feature and refines the object type based on the sensor output. To obtain an optimal snapshot, the feature should always be placed at the same distance from the camera and offer a surface orthogonal to the camera's lens axis, no matter what the orientation of the camera is, see *Figure 4.9*. The instruction *TplaceObj* provides such a functionality. We simply determine the snapshot position in the camera coordinate system and model it as a camera's feature. Again, we come back to an insertion operation in which a part feature is 'inserted' into or 'placed on' the camera's feature. Consequently, we can parameterize the generic task-level instruction *OinsertPart.* Only one action needs to be specified: TactivateSensor. When the part has reached the snapshot position, a picture must be taken and analyzed. The command TactivateSensor takes as argument the sensor name, the method to trigger, and possibly an attribute name for error detection purposes. It fetches the method definition, executes it, and updates the world model upon reception of the sensor output.

Various optimizations can be added to the identification process. For example, features already checked and assembly points can be easily skipped.

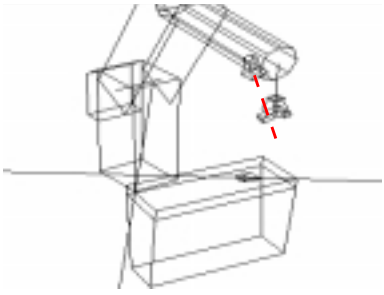


Figure 4.9: Object identification by a vision system

4.2.2 DISCUSSION

This example clearly shows the benefit of the two-layer task-level programming: generic and specialized. While the former synthesizes advanced and fully parameterized capabilities, the latter offers customized functionality for a particular application.

4.3 Force/Torque sensing

A force/torque sensor, mounted on the robot's wrist, can sense forces generated by the object being manipulated. Their main utilization is either to weight objects or to detect any contact with objects or surfaces [Gru94]. In the example given in this section, we describe the implementation of a guarded move. It typically consists of moving towards a supposed position until a force is felt. Such a function nowadays exists in robot languages, for example, Rapid's search function [Abb94}. However, they offer very limited tuning, and simply poll a given IO port for a binary value, thus making more elaborated data analysis not possible.

Another possible application, however not tested, is to correlate readings from a force sensor to detect part slippage during grasping.

4.3.1 OBJECT MODELING

The force/torque sensor used in this example does not send sensing output directly to the XPROB platform. It is instead connected to a force monitoring application running on a remote computer. As the physical implementation is not relevant for the application of our concern, we make abstractions of these considerations.

The modeling of the force/torque sensor is similar to the light beam specification presented in *Figure 3.4*. *Figure 4.10* depicts the status of the force sensor in the world model after a sensing operation. As indicated by the attribute *contact_RV*, the received value is 0 and this one is mapped into the logical value as false, as specified by the attribute *filter*.

class	Force/Torque	set
frame	world yxz 0 0 0 0 0 0	assumed
method	{rqst_contact RgetSensor 1000 force 1 "C 0"} {rqs	
hm		assumed
sensor_status	idle	checked
contact	false	checked
contact_RV	0	checked
access	TCP_IP 3700	assumed
filter	{C contact {false 0} {true 1}} {A {} }	
translation	{RgetSensor "$sensorCode"}	assumed

Figure 4.10: Force sensor description

4.3.2 HARDWARE DESCRIPTION

The force sensing system is connected to XPROB through an ethernet link, and communicates through the TCP/IP protocol, see *Figure 4.11*. The force/torque sensor itself is mounted on the robot wrist and connected to a computer running a LabView[1] application.

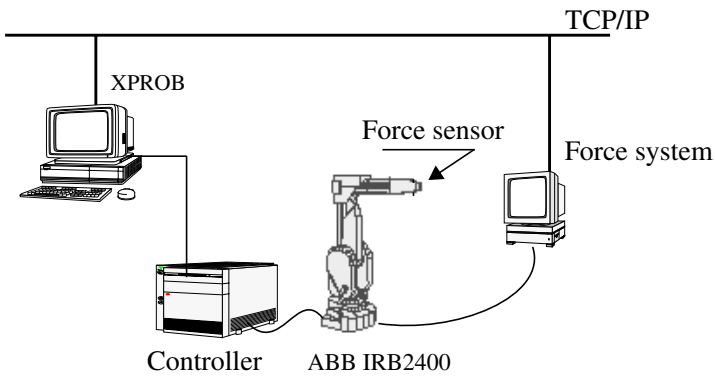


Figure 4.11: Force system application

1. LabView is a trademark of National Instruments Inc.

4.3.3 DESCRIPTION OF THE TASKS

The task Tsearch, detailed in *Figure 4.12*, simply refers to a guarded move operation. The values given as parameters indicate that the feature feat1 of the object *drill* must be put into contact with the feature feat1 of object *partToDrill,* the search starts at 30 mm from the actual position of the object *partToDrill*, and finally the contact is detected by the sensor *force.* The guarded move operation first initializes the sensor. It then approaches the assumed final position following a move-stop-sense sequence. Once the part's feature has been detected, the part's position is deduced to reflect the real location of the part in the current work-cell.

```
Task Tsearch {} {
    TguardedMove abb_robot drill feat1 partToDrill feat1 30 force
}
proc TguardedMove {robot tool feat_tool obj feat_obj initstep sensor } {
   #Initialize the sensor
   TactivateSensor $sensor rqst_reset
   set step $initstep
   # start the approach
   OmoveRelative $robot joint_move $tool $feat_tool $obj $feat_obj $step
   while {[TactivateSensor $sensor rqst_contact contact] == "false"} {
      incr step -3
      OmoveRelative $robot linear_move $tool $feat_tool $obj $feat_obj
$step
    }
    PdeducePos $robot $obj feature $feat_obj "" $step
}
```

Figure 4.12: Description of the search task

4.3.4 DISCUSSION

The example shows how sensing devices can be efficiently integrated into a robotic application. Our example suffers one major drawback, however, when executed in a real environment. It requires the robot manipulator to stop before each sensing. The values obtained from the sensor gain in accuracy, but this drastically increases the overall execution time of the task. Further improvements will therefore be needed to make it more valuable.

Chapter 5
Application to a robotized disassembly line

In this chapter, we describe an application of XPROB in a robotized work station for the disassembly of industrial asynchronous motors. Recycling electric motors is usually done using mechanical separation by shredding followed by magnetic separation. However, this method does not produce a high level of purity in the separated materials. Manual disassembly has proven to be unprofitable and hard labour. Robot-aided disassembly has therefore been investigated to offer an alternative solution to shredding.

Electric motors, such as washing-machine motors, have a rather simple construction principle and similar components, but there is a vast range of variation in design, depending on brand name and field of application. Because of these variations, the disassembly process requires an extensive use of sensors to get additional data about the parts to be manipulated. This implies a high flexibility from the disassembly program to take into account the sensor data and adapt the disassembly steps to the parts. For all these reasons, this project appears perfectly suitable for considering XPROB to control and program the robotized system at a high-level of specification .

5.1 Disassembly issues

While the area of automatic robotic assembly has been the focus of a great deal of research, little has been done regarding disassembly. As shown by an increasing number of recent projects [Sch99], [Dav99], [Hes99], this situation is about to change as environmental measures are constraining industry to recycle their end-of-life products. Automatic disassembly provides a cost effective means to dismantle products in clean material fractions that can then be recycled. Ideally disassembly could be treated as the inverse of assembly. However the problem is the variety of products and product states. The parts originate from various manufacturers and years of production. In addition, the items may not have been designed to be recycled. Finally, their shape or any feature is likely to be damaged.

5.2 Project description

Electric motors contain an important amount of copper that must be separated from the other materials in the motor to be efficiently recycled. Currently, less than 8% of the motors' copper can be recycled after shredding and magnetic separation. To understand the problematic of a motor's disassembly process, we first need to look at the motor structure as depicted in *Figure 5.1*. As previously mentioned, the motors have a rather simple structure. Two shields protect the stator, which rotates around an axis, the rotor. The stator, which is split in two pieces in *Figure 5.1*, contains the copper windings to be extracted. Within the scope of the REM-PRODUSE project, Karlsson [Kar97] describes a general approach for the disassembly of end-of-life electric motors. Firstly, the motor must be identified and its pose estimated through the use of a vision system. Then, the upper shield's screws must be unscrewed so that the shield and then the stator can be extracted from the rest of the motor. The next step is to cut the stator in half. Afterwards, a hydraulic tractive system pulls out the stator windings from the stator. Finally, the stator windings must be checked to guarantee that all copper has been removed. On the whole, four steps are

required to extract the copper from the motor. The next sections describe more thoroughly the different tasks and the implementation of the disassembly station.

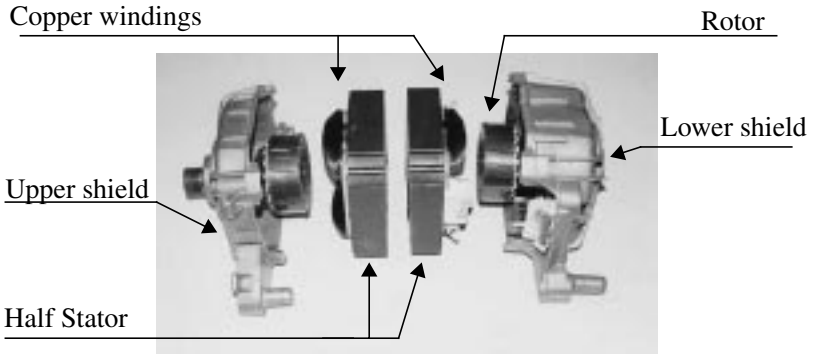


Figure 5.1: View of the structure of an electric motor

5.3 World Model description

In this section, the manipulated parts and the workcell's hardware are successively presented and their modeling described.

The motor and its constituent elements are modeled as *Part*, see "Object Classification" , section 3.2.1. *Figure 5.2* illustrates the existing modeling in the World Model of the various elements depicted in *Figure 5.1*. In this snapshot we can first identify the main elements, such as the rotor, stator, etc.. Their inherited classes have a suffix in the format of *_X,* and they specify a particular type. No direct relationship among parts can be deduced from this name convention. For example, a motor 'Motor_A' can be made up of a 'Upper_shield_C' and a 'Lower_shield_B'. Such a relationship is described in the section "Assembly & disassembly modeling" , section 4.1.1. A 'disassembly' method is added to each element that is to

be disassembled. The parameters of the method explicitly define the two resulting parts and their features in contact, depicted as follows:

```
disassemble Motor_A Upper_shield_C feat1 Lower_shield_B feat1
```

In the current implementation of XPROB, a part can be disassembled into only two pieces. A series of disassemblies leads, therefore, to a binary tree structure, depicted in *Figure 5.6*.

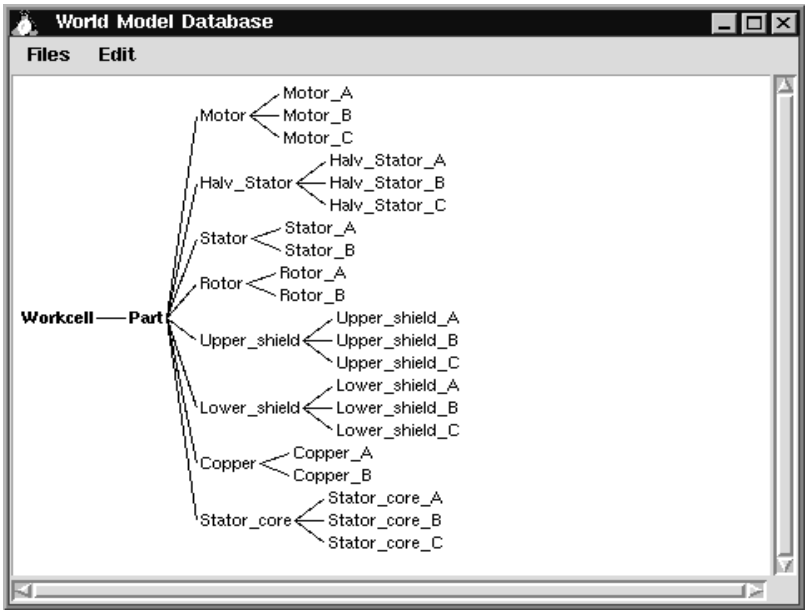


Figure 5.2: Motor parts specification

A great deal of hardware was required to implement our disassembly line. The snapshot, *Figure 5.3*, depicts the classes and instances that represent the equipment set-up in our workcell.

Controller: the Aramis software [Lob95] provides comprehensive tools to manage the disassembly station at a high level of abstraction. Preliminary integration tests were made, but a full integration was not possible in

our time frame. Consequently, the Aramis controller was not used in the project's evaluation phase.

Robot: an industrial robot, namely ABB IRB2400, was connected to an ABB S4 controller. The test-bed configuration presented in "Hardware description" , section 4.1.2 was reused. This configuration allows the robot to execute the commands on-the-fly

Tools: a specific gripper was designed to handle certain types of motor. Ideally several grippers should be available and an attribute for the gripper's shape should be associated to each motor so that the program can use the right tool and perform a safe and optimal grasping. A pneumatic drill can also be mounted on the end-effector and be activated by a digital signal.

Feeders (in/out): three conveyors were modeled either to bring the motors into the work cell, transport the shields for recycling, or convey the extracted copper to the proper bin. Four bins were also used to receive the end-products (stator_clean, stator_manual), the copper (bin_copper), and the shields (bin_shield).

Devices: an electric saw was located in a remote workshop for practical and safety reasons. This device was therefore only simulated at run-time. It was also the case for the hydraulic tractive system, which was out of order during the project's evaluation phase. However, it was possible to control this device using a small set of digital signals.

Fixtures: an in-house clamping device was used to keep the motor in a fixed position. A digital signal and a temporizer were needed to safely control it. A tool magazine was built to allow tool switching during the disassembly process.

Sensors: a vision system, based on a frame grabber [Nat97], a LabView[1] application and three DCC cameras, was set up and ultimately connected to XPROB. A force system, based on LabView and using a force sensor mounted on the robot wrist, could also communicate with XPROB. Two simple switches (light_beam, cut_done) were directly connected to the robot controller and were modeled to simulate presence detection sensors.

1. LabView is a trademark of National Instruments Inc.

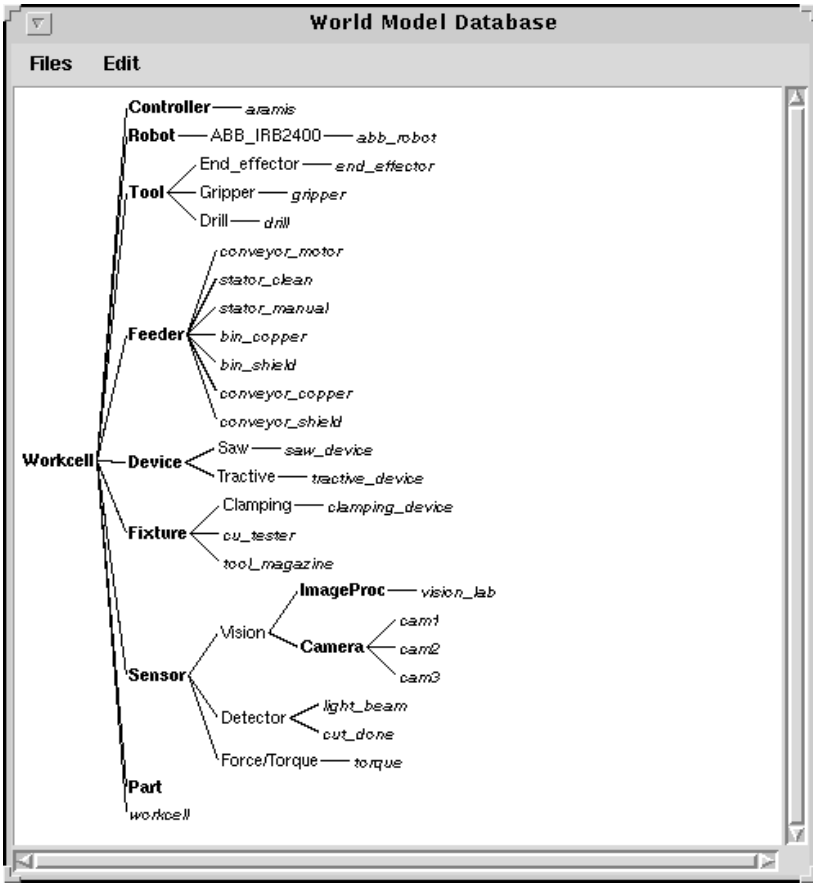


Figure 5.3: Partial World Model representation

5.4 Hardware description

Figure 5.4, below illustrates the physical implementation of the disassembly line.

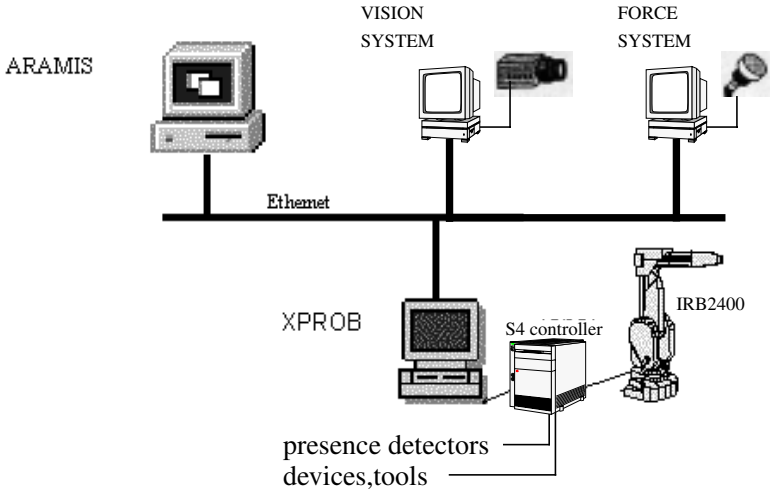


Figure 5.4: Hardware integration

5.5 Task-level programming of the application

A variable number of steps in the disassembly process can be pointed out, depending on the level of abstraction and complexity. In this section, we break down the process into 6 sequential steps that highlight the work done in each work station.

5.5.1 TASK SPECIFICATIONS

Sensor feedback is used at different stages during the disassembly, either to get more accurate information about the object or to update the object position/orientation. The actions performed at the different stations

are successively detailed in this section. The overall process is illustrated in *Figure 5.5*.

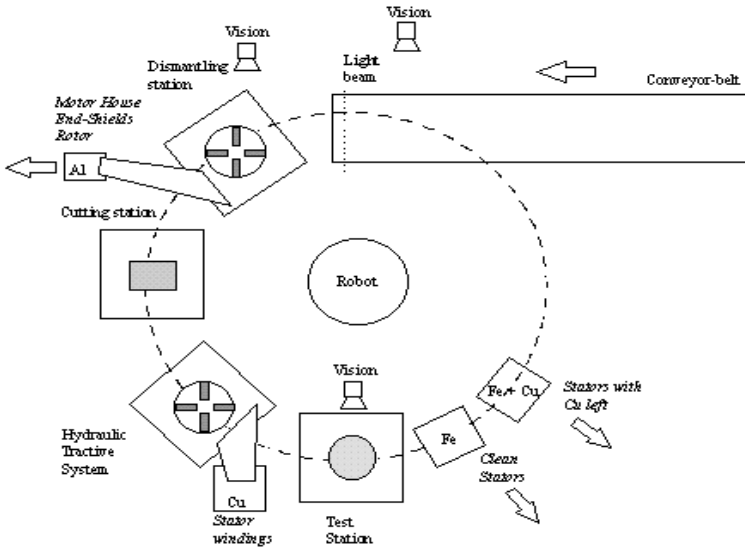


Figure 5.5: Overview of the disassembly line

Part I, Motor identification

Location: *In-feeder.*

Purpose: the goal is to try to identify the incoming motor and remove it from the conveyor belt.

Description: A motor arrives on the conveyor belt on a pre-defined side to simplify and optimize the recognition process. A light beam sensor at the end of the conveyor belt detects the motor. A picture is snapped by the camera and the frame grabber computes different parameters such as the object's size and a possible gripping position. Based on this position, we deduce the motor's center point. This information is then used to get the first list of potentially matching motor types. If this operation succeeds, the motor can be grasped and removed from the conveyor belt.

Part II, Dismantling

Location: *Dismantling station.*
Purpose: This is a transition process that puts the motor in a fixed position.
Description: The program computes the optimal object placement in the clamping device. It triggers the clamping device's opening, executes the motor's release, and executes another set of instructions to firmly clamp the motor.

Part III, Screw removal

Locations: *Dismantling station, tool magazine.*
Purpose: In this part, we identify the position of the screws that binds the upper- and lower-shields together, and drill the heads away.
Description: We obtain the screws' position using the vision sensor to locate them (x,y coordinates). If the vision system fails to return correct values, or if no motor in the database matches the number of screws detected, the motor is removed from the clamping device by the robot and placed back on the conveyor belt for manual classification. Otherwise, the robot places the gripper in the tool magazine and attaches the drill to the end-effector. The pneumatic drill is then placed above the first screw. The clamping device often lifts the motor up, thus making the screw head's position along the Z-axis in the World Model unreliable. Hence, a guarded move using the force sensor detects the correct position of the screw's head. The drill is then activated to shred it off. A similar sequence of actions is executed for the other screws. Upon completion, the robot returns the drill to the tool magazine and puts the gripper back on the end-effector.

Part IV, DismantleMotor
Location: *Dismantling station*.
Purpose: A sequence of mechanical operations extract the different motor parts.

Description: The robot successively extracts the upper-shield, the stator, and the lower-shield from the motor. The stator is placed at an electric saw, whereas the shields are dropped into a dedicated bin for recycling.

Part V, SawMotor

Location: *Cutting station.*
Purpose: The stator is sliced in two halves.
Description: Experimentation shows that the copper windings can be efficiently separated from the stator when it is cut in two halves, see *Figure 5.1*. The two halves are then placed on an out-feeder. Once the presence sensor detects the newly sliced stators, it triggers the next procedure: the copper extraction.

Part VI, RemoveCopper

Location: *Hydraulic tractive system.*
Purpose: This part consists of extracting as much copper as possible from the stator.
Description: When ready, a half-stator is gripped from the cutting station, placed and clamped in the hydraulic tractive system. Thereafter four hydraulic arms grip the windings and remove them from the stator. The system is pre-programmed, and only a few digital signals are necessary to control the different hydraulic commands.

Part VII, TestStator

Location: *Testing station.*
Purpose: An eddy current probe and a vision system check that all copper material has been removed.
Description: The stator is first placed on a table with an illuminating surface. Two pictures are successively snapped to detect copper colour and the number of empty stator slots. If any copper is found or if the number of slots does not match the expected value, the stator is assigned for manual removal of the copper. If this is not the case, the stator is rotated around an

eddy current probe to ensure that no copper remains. At the end of the testing process, the robot grips the stator and puts it in the appropriate bin.

Table 5.1 presents the execution time for each task and the total execution time. The data is provided by the XPROB's execution time manager. The estimated time is obtained by adding the worst-case execution time of all of the robot commands when successively triggered. The effective time corresponds to the recorded execution time of those commands.

Table 5.1: Tasks execution time

Task	**Estimated time (s)**	**Effective time (s)**
FetchMotor	36.50	16.17
ClampMotor	40.00	22.90
ScrewsShredding	272.00	149.90
DismantleMotor	188.00	76.20
SawMotor	2.00	1.90
RemoveCopper	99.00	45.08
TestStator	151.00	79.28
BackConveyor	20.00	19.84
TotalCycle	808.50	411.28

5.5.2 OBJECT REFINEMENT

So far we have assumed that the manipulated objects are clearly identified. This makes the overall process easier to explain, but does not match reality. Due to sensor approximation of, for example, the motor size, there is not enough information for the system to single out a unique motor. Therefore, XPROB creates a generic part, which is then refined along the disassembly process. In fact, the part is refined each time the force and vision sensors are used to sense the outside world. When a part has been clearly identified, its identity is propagated upwards to the parent parts to maintain database consistency.

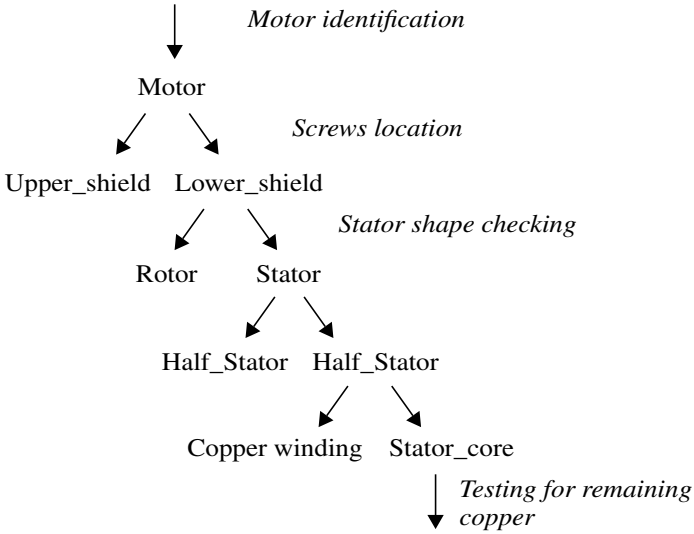


Figure 5.6: Sensor-based object refinement

5.6 Project evaluation

The programming and control of the disassembly line has allowed us to evaluate a research concept in a real environment. Its main contribution has been to highlight the assets and deficiencies of XPROB. The main objectives of XPROB (dynamic object-oriented modeling, high-level programming environment, and sensor integration) have been successfully achieved. On the other hand, the platform usability could be enhanced to ease object modeling and provide a more friendly program execution environment. It has also brought up new issues that were hidden in the simulation environment. Communication latency of the serial communication and the slow processing time of the robot controller led to jerky robot motion. In addition, inaccurate robot calibration, and mechanical deficiencies of the hardware in the workcell were among the most time-consuming and difficult problems to deal with.

Chapter 6
Conclusion

6.1 Summary and conclusions

Automated robotic applications require a flexible and open environment to perform advanced processes. In this thesis we have presented XPROB, an experimental platform for the development and control of robotic applications. Our software platform has been designed around an open architecture allowing customizing and enhanced connectivity. It also hides the complexity of robot programming from the user through the use of object-orientation and symbolic reasoning about the objects. The benefits of the XPROB platform are:

- High-level language for the task specification
- Object-oriented world model
- Sensor integration at runtime
- Support for modeling of 3-D objects
- Support for partially identified objects and dynamic refinement

Though the platform is by no means a complete system, the preliminary results of my research has lead to two major conclusions. Firstly, an

object-oriented approach bound to a task-level programming environment, presented in Section 3.2 and Section 3.3 respectively, efficiently provides a high flexibility and reusability during the modeling and programming phases. Secondly, an architecture that carries away the traditional off-line/ on-line programming distinction can considerably speed up the prototyping process. As explained in Section 3.7, we can take advantage of off-line simulation combined with on-line sensor feedback and program adaptation.

XPROB presents an approach to integrate heterogeneous commercial and research tools in a distributed environment. The evaluation of XPROB through the set-up of a disassembly line for electric motors demonstrates that a robotic system can efficiently make use of external sensor systems and devices. However, due to communication latency and time overhead, the ideas and techniques developed in this thesis cannot be applied to closed-loop control systems.

6.2 Future work

The intention during XPROB's design was to keep the focus on the platform architecture, the task-level language, and the object modeling. In addition, it was meant to provide foundations for further research in other domains. These axes of research encompass:

- Automatic grasp planning. This would be an interesting extension. Some research results [Jon90] could be integrated to generate an even higher level of automation.
- Gross motion planning. Further work has to be done to enhance the gross motion planner with a collision avoidance algorithm, as discussed in Section 3.4.2.
- Task specification. XPROB does not provide a higher level of abstraction for the task specification. Interfaces to more advanced programs, such as Aramis [Lob94], have been designed for this purpose. Nevertheless, this approach has been insufficiently tested, and more thorough investigation is needed.

Appendix

BNF DEFINITION OF THE WORLD-MODEL

Workcell::=
WORKCELL *workcell-id*
class Workcellset
robots *Robots-list* **set**
sensors *Sensors-list* **set**
tools *Tools-list* **set**
devices *Devices-list* **set**
feeders *Feeders-list* **set**
fixtures *Fixtures-list* **set**

Tools-list ::= {*tool-id*}*
Robots-list ::= {*robot-id*}*
Sensors-list ::= {*sensor-id*}*
Fixtures-list ::= {*fixture-id*}*
Feeders-list ::= {*feeder-id*}*
Devices-list ::= {*device-id*}*

Robot ::=
ROBOT *robot-id*
class Robot set
end_effector *end_effector-id* **set**

kinematics *Kinematics-params* **set**
motion *Motion-params* **set**
status *status-value conf-value*
access *Access-type* **set**
cad *Cad-data* **set**

Tool ::=
TOOL *tool-id*
class Tool set
frame *Frame-expression conf-value*
status *status-value conf-value*
access access-type **set**
tool_status tool_status-type **set**
method *Method-list* **set**
cad *Cad-data* **set**

End_effector ::=
END-EFFECTOR *end_effector-id*
class End_effector set
spec Tool set
frame *Frame-expression conf-value*
tool_attached *actuator-id* **set**
status *status-value conf-value*
tool_status *tool_status-typeconf-value*
access *Access-type* **set**
method *Method-list* **set**
cad *Cad-data* **set**

Sensor ::=
SENSOR *tool-id*
class Sensor set
frame *Frame-expression conf-value*
status *status-value conf-value*
[Sensor-attribute-list]

method *Method-list* **set**
access *Access-type* **set**
cad *Cad-data* **set**

Sensor-attribute-list ::=
Sensor-attribute {*Sensor-attribute}**

Sensor-attribute ::=
attribute-name-rv sensor-real-value conf-value
attribute-name-sv sensor-symb-value conf-value

Position ::=
POSITION position-id
class Position set
frame *Frame-expression* **set**
approach *Approach-expression* **set**
[**actuator_rotation** *rotation-value* **set**]

Camera ::=
CAMERA *camera-id*
class Camera set
frame *Frame-expression* **set**
cad *Cad-data* **set**

Gripper ::=
GRIPPER *gripper-id*
class Gripper set
type *gripper-type conf-value*
dimension *Dimension-expression* **set**
station *station-id conf-value*
frame *Frame-expression conf-value*
gripzone *Gripzone-list* **set**
grip_active *gripzone-id conf-value*
[**feature** *Feature-list* **set**]

object_gripped *object-id conf-value*
gripper_status G*ripper-status-value conf-value*
access *Access-type* **set**
method *Method-list* **set**
cad *Cad-data* **set**

Drill ::=
DRILL drill-id
class Drill set
type *drill-type conf-value*
dimension *Dimension-expression* **set**
station *station-id conf-value*
frame *Frame-expression conf-value*
gripzone *Gripzone-list* **set**
grip_active *Gripzone-id conf-value*
[**feature** *Feature-list* **set**]
object_gripped *object-id conf-value*
drill_status *drill-status-value conf-value*
access *Access-type* **set**
method *Method-list* **set**
cad *Cad-data* **set**

Part ::=
PART *part-id*
class Part set
dimension *Dimension-expression* **set**
station *station-id conf-value*
frame *Frame-expression conf-value*
gripzone *Gripzone-list* **set**
grip_active *gripzone-id conf-value*
[**feature** *Feature-list* **set**]
method *Method-part-list* **set**
madeup *Madeup-expression conf-value*
[*Part-attribute-list*]
cad *Cad-data* **set**

Part-attribute-list ::=
    {*attribute-name attribute-value conf-value*}*

Part-type-list ::=
    Part-type-id {*Part-type-id*}*

Part-type-id ::=
    Part-class-id _ Code

Part-id ::=
    Part-type-id _ Part-counter

Frame-expression ::=
    coordinate-system-id Orientation-expression
    Position-expression

Orientation-expression ::=
    yxz *alpha-value beta-value gamma-value*

Position-expression ::=
    x-value y-value z-value

Gripzone-list ::=
    gripzone-expression { *gripzone-expression*}*

Gripzone-expression ::=
    coordinate-system-id Orientation-expression
    Position-expression zone-name gripper-rotation

Feature-list ::=
    Feature-expression { *Feature-expression*}*

Feature-expression ::=
coordinate-system-id Orientation-expression
Position-expression feature-id feature-type-id
Dimension-expression

Dimension-expression ::=
length-value width-value height-value

Method-list ::=
Method-expression {*Method-expression*}*

Method-expression ::=
method-name function-name parameters-list

Access-type ::=
TCP/IP *port-number* **| COM** *port-number*

Status-value ::=
run | stop | error

Cad-data ::=
*Cad-name { Cad-parameter }**

Conf-value ::=
assumed | unknown | set | checked | computed

References

[Abb94] ABB Flexible Automation AB, Rapid Reference Manual 3.0

[And97] Andeen, G.N., Toward a science of assembly, Robotics and Autonomous Ssytems, vol. 21, 1997, pp. 240-248.

[Bar95] Baartman, J.P., Automation of assembly operations on parts, Delft University, PhD Thesis, 1995.

[Dac92] DaCosta, F., Hwang, V., Khosla, P., Lumia, R., An integrated prototyping environment for programmable automation, SPIE/OE 92 International Symposium on Intelligent Robot in Space.

[Dav99] David, B.T., Boutros, N., Saikali, K., Chevron, D., Gerner, S., Skaf, A., Binder, Z., Dubois, M., Automation of disassembly processes and its information systems, Proceedings of First International Symposium On Environmentally Conscious Design and Inverse Manufacturing, 1999, pp. 564-569.

[Fah98] Fahim, A., Choi, K., The Uniset approach for the programming of flexible manufacturing cell, Robotics and Computer-Integrated Manufacturing, Elsevier, 1998, #14. pages 69-78.

[Fle94] Fleury, S., Herrb, M., Chatila, R., Design of a modular architecture for autonomous robot, Proceedings of IEEE International Conference on Robotics and Automation 1994, pp. 3508-3513.

[Fun90] Funda, J., Taylor, R.H., Paul, R., On Homogeneous Transforms, Quaternions, and Computational Efficiency, IEEE International Conference on Robotics and Automation 1990, Vol6, no 3, pp. 382-388.

[Gra89] Gray, J.J, Introduction to robotics, Second Edition, Addison-Wesley Publishing Company, ISBN 0-201-09528-9.

[Gru94] Gruver, W.A., Intelligent Robotics in Manufacturing, Service, and Rehabilitation: an overview, IEEE Transactions on Industrial Electronics, vol. 41, no 1, pp. 4-11.

[Hea86] Hearn, D., Baker, M., Computer Graphics. Prentice-Hall Publisher, 1986. Chap. 10, Three-dimensional object representations.

[Hes99] Hesselbach, J., Westernhagen, K. v., Disassembly simulation for an effective recycling of electrical scrap, Proceedings of First International Symposium On Environmentally Conscious Design and Inverse Manufacturing, 1999, pp. 582-585.

[Hua97] Huaguo, L., Cuiyun, J., Jianan, H., A knowledge-based approach for object classification for robotic assembly, IEEE International Conference on Intelligent Processing Systems, 1997, pp. 1260-1262.

[Hwa92] Hwang, Y.K., Ahuja, N., Gross motion planning: a survey. ACM Computing Surveys, Vol 24. No 3. September 1992.

[Hwa96] Hwang, C.P., Ho, C.S., A knowledge-based task-level programming and execution environment for robots, Robotics and Computer-Integrated manufacturing, vol. 12 1996, no 4; pp. 329-351.

[Kar97] Karlsson, B., Karlsson, N., Lauber, A., Sensor system for disassembly of electrical motor, Proceedings of Robotikdagarna, 1997, pp. 31-40.

[Jon90] Jones, J.L., Lozano-Perez, T., Planning two-fingered grasps for pick-and-place operations on polyhedra, Proceedings of IEEE International Conference on Robotics and Automation, 1990, pp. 683-688.

[Lap99] Lapham, J., RobotScript: the introduction of a universal robot programming language, Industrial Robot: An International Journal, vol. 26, 1999, pp. 17-25.

[Lau90] Laugier, C., Ijel, A., Troccaz. J., Combining vision based information and partial geometric models in automatic grasping. IEEE International Conference on Robotics and Automation 1990

[Li95] Li, T.S., Latombe, J.C., On-line Manipulation Planning for two Robot Arms in a Dynamic Environment, IEEE International Conference on Robotics and Automation 1995.

[Lob94] Loborg, P., Holmbom, P., Sköld, M., Törne,. A., A model for the execution of task level specifications for intelligent and flexible manufacturing systems.

[Lob95] Loborg P., Törne, A., A layered Architecture for Real-Time Applications. Euromicro Conference on Real-Time Systems, Odense, Denmark, 1995.

[Loz89] Lozano-Perez, T., Jones, J.L., Mazer, E., O'Donnell, P.A., Task-level planning of pick-and-place robot motions, Computer, #22 3, March 1989 , pp. 21 -29.

[Mak99] Mak, K.L., Lau, H.Y.K, Wong, S.T.W., Object-oriented specification of automated manufacturing systems. Robotics and Computer-Integrated Manufacturing, Elsevier, 1999, #15. pages 297-312.

[Mil91] Miller, D.J., Lennox, R.C., An object-oriented environment for robot system architecture, IEEE Control Systems. Pages 14- 23

[Muj82] Mujtaba, M.S., Goldman, R., Binford, T., Stanford's AL Robot Programming Language, Computers in Mechanical Engineering, August 1982.

[Nna93] Nnaji, B.O., Theory of automatic robot assembly and programming. Chapman & Hall. ISBN 0-412-39310-7, 1993.

[Nat97] National Instruments Corporation, Imaq Vision for G Reference Manual, June 1997

[Nis98] Nishiyama, H., Ohwada, H., Mizoguchi, F., A multiagent robot language for communication and concurrency control, Proceedings of International Conference on Multi Agent Systems, 1998, pp. 206-213.

[Par95] Pardo-Castellote, G., Schneider, S.A., Cannon Jr, R.F.,. System Design and Interfaces for intelligent manufacturing workcell, Proceedings of the IEEE International Conference on Intelligent Robots and Systems, Nagoya, 1995.

[Pet96] Pettinaro, G.C., Basic Set of beaviors for programming assembly robots, Ph.D. thesis, University of Edinburgh.

[Pri93] Prinz, M., Liu, H.C., Nnaiji, B.O., From CAD-based kinematic modeling to automated robot programming. Technical report, 1993.

[Ren97] Renfors, J., Real-time teleoperation of ABB S4 robots, Proceedings of Robotikdagarna, 1997, section G, pp. 73-80.

[Rem93] Rembold et al, Computer Integrated Manufacturing and Engineering, Addison Wesley, 1993.

[Sch91] Schrott, G., An experimental environment for task-level programming of robots, In 2nd Int. Symposium on Experimental Robotics, Toulouse, 1991, pp.196-206.

[Sch99] Scholz-Reiter, B., Scharke, H., Hucht, A., Flexible robot-based diassembly cell for obsolete TV-sets and monitors, Robotics and Computer-Integrated manufacturing, vol. 15 1999, pp. 247-255.

[Smi96] Smith, C.E., Papanikolopoulos, N.P., Vision-guided robotic grasping: Issues and Experiments, Proceedings of the 1996 IEEE International Conference on Robotics and Automation, pp. 3203-3208.

[Tay82] Taylor, R.H., Summers, P.D., Meyer, J.M., AML: a manufacturing language, The International Journal of Robotics Research, vol. 1, No 3, 1982.

[Tun94] C. P. Tung, A. C. Kak. Integrating Sensing, task planning, and execution, Proceedings of the 1996 IEEE International Conference on Robotics and Automation, pp. 2030-2037.

[Wit95] Wittenberg, G., Developments in offline programming: an overview, Industrial Robot, An International Journal, vol. 22, 1995, pp. 21-23.

Avdelning, Institution
Division, department

Department of Computer and Information Science

Institutionen för datavetenskap

Datum
Date

April 2000

Språk
Language

☐ Svenska/Swedish
☒ Engelska/English
☐ _____

Rapporttyp
Report: category

☒ Licentiatavhandling
☐ Examensarbete
☐ C-uppsats
☐ D-uppsats
☐ Övrig rapport
☐ _____

ISBN 91-7219-701-3

ISRN LiU-Tek-Lic-2000:16

Serietitel och serienummer
Title of series, numbering

ISSN 0280-7971

Linköping Studies in Science and Technology

Thesis No. 820

URL för elektronisk version

Titel
Title

Control of industrial robots through high-level task programming

Författare
Author

Jean Paul Meynard

Sammandrag
Abstract

In this thesis we present an experimental research platform in robotics, XPROB. This platform has been designed to be a tool that facilitates the development of robotic applications. XPROB achieves a flexible prototyping system that features a task-level programming environment, a dynamic representation of the work-cell's equipment, and sensor data integration at runtime allowing on-line program monitoring and adaptation.

This thesis describes how the object-orientation paradigm combined with a traditional layered-control structure lead to an open and dynamic architecture. It also presents an advanced object representation to handle high-level reasoning, even about partially recognized objects.

The platform was first evaluated using simple robotic applications, such as assembly and sensor-guided actions. Afterwards, an industrial application, consisting of a disassembly line for worn-out electric motors, was successfully set up and controlled by our platform.

This work has been supported by the Swedish National Board for Industrial and Technical Development (NUTEK).

Nyckelord
Keywords

Task-level programming, Industrial robot, Robotic platform, On-line programming, Object-orientation

Department of Computer and Information Science
Linköpings universitet

Linköping Studies in Science and Technology

No 17 **Vojin Plavsic:** Interleaved Processing of Non-Numerical Data Stored on a Cyclic Memory. (Available at: FOA, Box 1165, S-581 11 Linköping, Sweden. FOA Report B30062E)

No 28 **Arne Jönsson, Mikael Patel:** An Interactive Flowcharting Technique for Communicating and Realizing Algorithms, 1984.

No 29 **Johnny Eckerland:** Retargeting of an Incremental Code Generator, 1984.

No 48 **Henrik Nordin:** On the Use of Typical Cases for Knowledge-Based Consultation and Teaching, 1985.

No 52 **Zebo Peng:** Steps Towards the Formalization of Designing VLSI Systems, 1985.

No 60 **Johan Fagerström:** Simulation and Evaluation of Architecture based on Asynchronous Processes, 1985.

No 71 **Jalal Maleki:** ICONStraint, A Dependency Directed Constraint Maintenance System, 1987.

No 72 **Tony Larsson:** On the Specification and Verification of VLSI Systems, 1986.

No 73 **Ola Strömfors:** A Structure Editor for Documents and Programs, 1986.

No 74 **Christos Levcopoulos:** New Results about the Approximation Behavior of the Greedy Triangulation, 1986.

No 104 **Shamsul I. Chowdhury: S**tatistical Expert Systems - a Special Application Area for Knowledge-Based Computer Methodology, 1987.

No 108 **Rober Bilos:** Incremental Scanning and Token-Based Editing, 1987.

No 111 **Hans Block:** SPORT-SORT Sorting Algorithms and Sport Tournaments, 1987.

No 113 **Ralph Rönnquist:** Network and Lattice Based Approaches to the Representation of Knowledge, 1987.

No 118 **Mariam Kamkar, Nahid Shahmehri:** Affect-Chaining in Program Flow Analysis Applied to Queries of Programs, 1987.

No 126 **Dan Strömberg:** Transfer and Distribution of Application Programs, 1987.

No 127 **Kristian Sandahl:** Case Studies in Knowledge Acquisition, Migration and User Acceptance of Expert Systems, 1987.

No 139 **Christer Bäckström:** Reasoning about Interdependent Actions, 1988.

No 140 **Mats Wirén:** On Control Strategies and Incrementality in Unification-Based Chart Parsing, 1988.

No 146 **Johan Hultman:** A Software System for Defining and Controlling Actions in a Mechanical System, 1988.

No 150 **Tim Hansen:** Diagnosing Faults using Knowledge about Malfunctioning Behavior, 1988.

No 165 **Jonas Löwgren:** Supporting Design and Management of Expert System User Interfaces, 1989.

No 166 **Ola Petersson:** On Adaptive Sorting in Sequential and Parallel Models, 1989.

No 174 **Yngve Larsson:** Dynamic Configuration in a Distributed Environment, 1989.

No 177 **Peter Åberg:** Design of a Multiple View Presentation and Interaction Manager, 1989.

No 181 **Henrik Eriksson:** A Study in Domain-Oriented Tool Support for Knowledge Acquisition, 1989.

No 184 **Ivan Rankin:** The Deep Generation of Text in Expert Critiquing Systems, 1989.

No 187 **Simin Nadjm-Tehrani:** Contributions to the Declarative Approach to Debugging Prolog Programs, 1989.

No 189 **Magnus Merkel:** Temporal Information in Natural Language, 1989.

No 196 **Ulf Nilsson:** A Systematic Approach to Abstract Interpretation of Logic Programs, 1989.

No 197 **Staffan Bonnier:** Horn Clause Logic with External Procedures: Towards a Theoretical Framework, 1989.

No 203 **Christer Hansson:** A Prototype System for Logical Reasoning about Time and Action, 1990.

No 212 **Björn Fjellborg:** An Approach to Extraction of Pipeline Structures for VLSI High-Level Synthesis, 1990.

No 230 **Patrick Doherty:** A Three-Valued Approach to Non-Monotonic Reasoning, 1990.

No 237 **Tomas Sokolnicki:** Coaching Partial Plans: An Approach to Knowledge-Based Tutoring, 1990.

No 250 **Lars Strömberg:** Postmortem Debugging of Distributed Systems, 1990.

No 253 **Torbjörn Näslund:** SLDFA-Resolution - Computing Answers for Negative Queries, 1990.

No 260 **Peter D. Holmes:** Using Connectivity Graphs to Support Map-Related Reasoning, 1991.

No 283 **Olof Johansson:** Improving Implementation of Graphical User Interfaces for Object-Oriented Knowledge-Bases, 1991.

No 298 **Rolf G Larsson:** Aktivitetsbaserad kalkylering i ett nytt ekonomisystem, 1991.

No 318 **Lena Strömbäck:** Studies in Extended Unification-Based Formalism for Linguistic Description: An Algorithm for Feature Structures with Disjunction and a Proposal for Flexible Systems, 1992.

No 319 **Mikael Pettersson:** DML-A Language and System for the Generation of Efficient Compilers from Denotational Specification, 1992.

No 326 **Andreas Kågedal:** Logic Programming with External Procedures: an Implementation, 1992.

No 328 **Patrick Lambrix:** Aspects of Version Management of Composite Objects, 1992.

No 333 **Xinli Gu:** Testability Analysis and Improvement in High-Level Synthesis Systems, 1992.

No 335 **Torbjörn Näslund:** On the Role of Evaluations in Iterative Development of Managerial Support Sytems, 1992.

No 348 **Ulf Cederling:** Industrial Software Development - a Case Study, 1992.

No 352 **Magnus Morin:** Predictable Cyclic Computations in Autonomous Systems: A Computational Model and Implementation, 1992.

No 371 **Mehran Noghabai:** Evaluation of Strategic Investments in Information Technology, 1993.

No 378 **Mats Larsson:** A Transformational Approach to Formal Digital System Design, 1993.

No 380 **Johan Ringström:** Compiler Generation for Parallel Languages from Denotational Specifications, 1993.

No 381 **Michael Jansson:** Propagation of Change in an Intelligent Information System, 1993.

No 383 **Jonni Harrius:** An Architecture and a Knowledge Representation Model for Expert Critiquing Systems, 1993.

No 386 **Per Österling:** Symbolic Modelling of the Dynamic Environments of Autonomous Agents, 1993.

No 398 **Johan Boye:** Dependency-based Groudness Analysis of Functional Logic Programs, 1993.

No 402 **Lars Degerstedt:** Tabulated Resolution for Well Founded Semantics, 1993.
No 406 **Anna Moberg:** Satellitkontor - en studie av kommunikationsmönster vid arbete på distans, 1993.
No 414 **Peter Carlsson:** Separation av företagsledning och finansiering - fallstudier av företagsledarutköp ur ett agentteoretiskt perspektiv, 1994.
No 417 **Camilla Sjöström:** Revision och lagreglering - ett historiskt perspektiv, 1994.
No 436 **Cecilia Sjöberg:** Voices in Design: Argumentation in Participatory Development, 1994.
No 437 **Lars Viklund:** Contributions to a High-level Programming Environment for a Scientific Computing, 1994.
No 440 **Peter Loborg:** Error Recovery Support in Manufacturing Control Systems, 1994.
FHS 3/94 **Owen Eriksson:** Informationssystem med verksamhetskvalitet - utvärdering baserat på ett verksamhetsinriktat och samskapande perspektiv, 1994.
FHS 4/94 **Karin Pettersson:** Informationssystemstrukturering, ansvarsfördelning och användarinflytande - En komparativ studie med utgångspunkt i två informationssystemstrategier, 1994.
No 441 **Lars Poignant:** Informationsteknologi och företagsetablering - Effekter på produktivitet och region, 1994.
No 446 **Gustav Fahl:** Object Views of Relational Data in Multidatabase Systems, 1994.
No 450 **Henrik Nilsson:** A Declarative Approach to Debugging for Lazy Functional Languages, 1994.
No 451 **Jonas Lind:** Creditor - Firm Relations: an Interdisciplinary Analysis, 1994.
No 452 **Martin Sköld:** Active Rules based on Object Relational Queries - Efficient Change Monitoring Techniques, 1994.
No 455 **Pär Carlshamre:** A Collaborative Approach to Usability Engineering: Technical Communicators and System Developers in Usability-Oriented Systems Development, 1994.
FHS 5/94 **Stefan Cronholm:** Varför CASE-verktyg i systemutveckling? - En motiv- och konsekvensstudie avseende arbetssätt och arbetsformer, 1994.
No 462 **Mikael Lindvall:** A Study of Traceability in Object-Oriented Systems Development, 1994.
No 463 **Fredrik Nilsson:** Strategi och ekonomisk styrning - En studie av Sandviks förvärv av Bahco Verktyg, 1994.
No 464 **Hans Olsén:** Collage Induction: Proving Properties of Logic Programs by Program Synthesis, 1994.
No 469 **Lars Karlsson:** Specification and Synthesis of Plans Using the Features and Fluents Framework, 1995.
No 473 **Ulf Söderman:** On Conceptual Modelling of Mode Switching Systems, 1995.
No 475 **Choong-ho Yi:** Reasoning about Concurrent Actions in the Trajectory Semantics, 1995.
No 476 **Bo Lagerström:** Successiv resultatavräkning av pågående arbeten. - Fallstudier i tre byggföretag, 1995.
No 478 **Peter Jonsson:** Complexity of State-Variable Planning under Structural Restrictions, 1995.
FHS 7/95 **Anders Avdic:** Arbetsintegrerad systemutveckling med kalkylkprogram, 1995.
No 482 **Eva L Ragnemalm:** Towards Student Modelling through Collaborative Dialogue with a Learning Companion, 1995.
No 488 **Eva Toller:** Contributions to Parallel Multiparadigm Languages: Combining Object-Oriented and Rule-Based Programming, 1995.
No 489 **Erik Stoy:** A Petri Net Based Unified Representation for Hardware/Software Co-Design, 1995.
No 497 **Johan Herber:** Environment Support for Building Structured Mathematical Models, 1995.
No 498 **Stefan Svenberg:** Structure-Driven Derivation of Inter-Lingual Functor-Argument Trees for Multi-Lingual Generation, 1995.
No 503 **Hee-Cheol Kim:** Prediction and Postdiction under Uncertainty, 1995.
FHS 8/95 **Dan Fristedt:** Metoder i användning - mot förbättring av systemutveckling genom situationell metodkunskap och metodanalys, 1995.
FHS 9/95 **Malin Bergvall:** Systemförvaltning i praktiken - en kvalitativ studie avseende centrala begrepp, aktiviteter och ansvarsroller, 1995.
No 513 **Joachim Karlsson:** Towards a Strategy for Software Requirements Selection, 1995.
No 517 **Jakob Axelsson:** Schedulability-Driven Partitioning of Heterogeneous Real-Time Systems, 1995.
No 518 **Göran Forslund:** Toward Cooperative Advice-Giving Systems: The Expert Systems Experience, 1995.
No 522 **Jörgen Andersson:** Bilder av småföretagares ekonomistyrning, 1995.
No 538 **Staffan Flodin:** Efficient Management of Object-Oriented Queries with Late Binding, 1996.
No 545 **Vadim Engelson:** An Approach to Automatic Construction of Graphical User Interfaces for Applications in Scientific Computing, 1996.
No 546 **Magnus Werner :** Multidatabase Integration using Polymorphic Queries and Views, 1996.
FiF-a 1/96 **Mikael Lind:** Affärsprocessinriktad förändringsanalys - utveckling och tillämpning av synsätt och metod, 1996.
No 549 **Jonas Hallberg:** High-Level Synthesis under Local Timing Constraints, 1996.
No 550 **Kristina Larsen:** Förutsättningar och begränsningar för arbete på distans - erfarenheter från fyra svenska företag. 1996.
No 557 **Mikael Johansson:** Quality Functions for Requirements Engineering Methods, 1996.
No 558 **Patrik Nordling:** The Simulation of Rolling Bearing Dynamics on Parallel Computers, 1996.
No 561 **Anders Ekman:** Exploration of Polygonal Environments, 1996.
No 563 **Niclas Andersson:** Compilation of Mathematical Models to Parallel Code, 1996.
No 567 **Johan Jenvald:** Simulation and Data Collection in Battle Training, 1996.
No 575 **Niclas Ohlsson:** Software Quality Engineering by Early Identification of Fault-Prone Modules, 1996.
No 576 **Mikael Ericsson:** Commenting Systems as Design Support—A Wizard-of-Oz Study, 1996.
No 587 **Jörgen Lindström:** Chefers användning av kommunikationsteknik, 1996.
No 589 **Esa Falkenroth:** Data Management in Control Applications - A Proposal Based on Active Database Systems, 1996.
No 591 **Niclas Wahllöf:** A Default Extension to Description Logics and its Applications, 1996.
No 595 **Annika Larsson:** Ekonomisk Styrning och Organisatorisk Passion - ett interaktivt perspektiv, 1997.
No 597 **Ling Lin:** A Value-based Indexing Technique for Time Sequences, 1997.

No 598 **Rego Granlund:** C^3Fire - A Microworld Supporting Emergency Management Training, 1997.
No 599 **Peter Ingels:** A Robust Text Processing Technique Applied to Lexical Error Recovery, 1997.
No 607 **Per-Arne Persson:** Toward a Grounded Theory for Support of Command and Control in Military Coalitions, 1997.
No 609 **Jonas S Karlsson:** A Scalable Data Structure for a Parallel Data Server, 1997.
FiF-a 4 **Carita Åbom:** Videomötesteknik i olika affärssituationer - möjligheter och hinder, 1997.
FiF-a 6 **Tommy Wedlund**: Att skapa en företagsanpassad systemutvecklingsmodell - genom rekonstruktion, värdering och vidareutveckling i T50-bolag inom ABB, 1997.
No 615 **Silvia Coradeschi**: A Decision-Mechanism for Reactive and Coordinated Agents, 1997.
No 623 **Jan Ollinen:** Det flexibla kontorets utveckling på Digital - Ett stöd för multiflex? 1997.
No 626 **David Byers:** Towards Estimating Software Testability Using Static Analysis, 1997.
No 627 **Fredrik Eklund:** Declarative Error Diagnosis of GAPLog Programs, 1997.
No 629 **Gunilla Ivefors:** Krigsspel coh Informationsteknik inför en oförutsägbar framtid, 1997**.**
No 631 **Jens-Olof Lindh:** Analysing Traffic Safety from a Case-Based Reasoning Perspective, 1997
No 639 **Jukka Mäki-Turja:**. Smalltalk - a suitable Real-Time Language, 1997.
No 640 **Juha Takkinen**: CAFE: Towards a Conceptual Model for Information Management in Electronic Mail, 1997.
No 643 **Man Lin**: Formal Analysis of Reactive Rule-based Programs, 1997.
No 653 **Mats Gustafsson**: Bringing Role-Based Access Control to Distributed Systems, 1997.
FiF-a 13 **Boris Karlsson:** Metodanalys för förståelse och utveckling av systemutvecklingsverksamhet. Analys och värdering av systemutvecklingsmodeller och dess användning, 1997.
No 674 **Marcus Bjäreland:** Two Aspects of Automating Logics of Action and Change - Regression and Tractability, 1998.
No 676 **Jan Håkegård**: Hiera rchical Test Architecture and Board-Level Test Controller Synthesis, 1998.
No 668 **Per-Ove Zetterlund**: Normering av svensk redovisning - En studie av tillkomsten av Redovisningsrådets rekommendation om koncernredovisning (RR01:91), 1998.
No 675 **Jimmy Tjäder**: Projektledaren & planen - en studie av projektledning i tre installations- och systemutvecklingsprojekt, 1998.
FiF-a 14 **Ulf Melin**: Informationssystem vid ökad affärs- och processorientering - egenskaper, strategier och utveckling, 1998.
No 695 **Tim Heyer**: COMPASS: Introduction of Formal Methods in Code Development and Inspection, 1998.
No 700 **Patrik Hägglund:** Programming Languages for Computer Algebra, 1998.
FiF-a 16 **Marie-Therese Christiansson:** Inter-organistorisk verksamhetsutveckling - metoder som stöd vid utveckling av partnerskap och informationssystem, 1998.
No 712 **Christina Wennestam:** Information om immateriella resurser. Investeringar i forskning och utveckling samt i personal inom skogsindustrin, 1998.
No 719 **Joakim Gustafsson:** Extending Temporal Action Logic for Ramification and Concurrency, 1998.
No 723 **Henrik André-Jönsson:** Indexing time-series data using text indexing methods, 1999.
No 725 **Erik Larsson:** High-Level Testability Analysis and Enhancement Techniques, 1998.
No 730 **Carl-Johan Westin:** Informationsförsörjning: en fråga om ansvar - aktiviteter och uppdrag i fem stora svenska organisationers operativa informationsförsörjning, 1998.
No 731 **Åse Jansson:** Miljöhänsyn - en del i företags styrning, 1998.
No 733 **Thomas Padron-McCarthy:** Performance-Polymorphic Declarative Queries, 1998.
No 734 **Anders Bäckström:** Värdeskapande kreditgivning - Kreditriskhantering ur ett agentteoretiskt perspektiv, 1998.
FiF-a 21 **Ulf Seigerroth:** Integration av förändringsmetoder - en modell för välgrundad metodintegration, 1999.
FiF-a 22 **Fredrik Öberg:** Object-Oriented Frameworks - A New Strategy for Case Tool Development, 1998.
No 737 **Jonas Mellin:** Predictable Event Monitoring, 1998.
No 738 **Joakim Eriksson:** Specifying and Managing Rules in an Active Real-Time Database System, 1998.
FiF-a 25 **Bengt E W Andersson:** Samverkande informationssystem mellan aktörer i offentliga åtaganden - En teori om aktörsarenor i samverkan om utbyte av information, 1998.
No 742 **Pawel Pietrzak:** Static Incorrectness Diagnosis of CLP (FD), 1999.
No 748 **Tobias Ritzau:** Real-Time Reference Counting in RT-Java, 1999.
No 751 **Anders Ferntoft:** Elektronisk affärskommunikation - kontaktkostnader och kontaktprocesser mellan kunder och leverantörer på producentmarknader,1999.
No 752 **Jo Skåmedal:** Arbete på distans och arbetsformens påverkan på resor och resmönster, 1999.
No 753 **Johan Alvehus:** Mötets metaforer. En studie av berättelser om möten, 1999.
No 766 **Martin V. Howard:** Designing dynamic visualizations of temporal data, 1999.
No 769 **Jesper Andersson:** Towards Reactive Software Architectures, 1999.
No 775 **Anders Henriksson:** Unique kernel diagnosis, 1999.
FiF-a 30 **Pär J. Ågerfalk:** Pragmatization of Information Systems - A Theoretical and Methodological Outline, 1999.
No 787 **Charlotte Björkegren:** Learning for the next project - Bearers and barriers in knowledge transfer within an organisation, 1999.
No 790 **Erik Berglund:** Use-Oriented Documentation in Software Development, 1999.
No 791 **Klas Gäre:** Verksamhetsförändringar i samband med IS-införande, 1999.
No 800 **Anders Subotic:** Software Quality Inspection, 1999.
No 809 **Flavius Gruian:** Energy-Aware Design of Digital Systems, 2000.
FiF-a 32 **Karin Hedström:** Kunskapsanvändning och kunskapsutveckling hos verksamhetskonsulter - Erfarenheter från ett FOU-samarbete, 2000.
No 808 **Linda Askenäs:** Affärssystemet - En studie om teknikens aktiva och passiva roll i en organisation, 2000.
No 820 **Jean Paul Meynard:** Control of industrial robots through high-level task programming, 2000.