A CLIENT–SERVER BASED SYSTEM FOR PROGRAMMING AND CONTROL OF COOPERATING ROBOTS

Boris Baginski, German Kess

Technische Universität München — Department of Robotics and Real-Time Systems

Abstract: In this paper, we present a concept to program and control a robot system consisting of several sensors, actors and manipulators. To integrate all these heterogenous components, we define general requirements for an execution layer, its internal structure and the services it has to provide for a higher level planning system or a user written robot application. We present our implemented execution layer, an extendable and powerful client-server system controlling two robots and multiple sensors and actors in one workcell.

Keywords: robot programming, cooperating robots, execution layer, control

1 Introduction

Our major research interests focus on the development of autonomous robot systems. These systems shall consist of several cooperating manipulators, and the user communicates with them in a task-level fashion. An example for this scenario would be to place a Rubic's Cube in the workcell and to enter the command *Solve the cube!* The autonomous system would then decompose this task, localize the cube, generate a set of operations from the sensed position and execute these operations on the cube. All steps would be supervised and guided by sensors to ensure proper operation and to react and replan in cases of errors.

To develop such a powerful multi-level planning and execution system, a wide range of hardware components has to be controlled in concert. Several of these components, especially the robots themselves, demand real-time control of poses, forces and torques. The sensors in general are not fixed to control one device, but will usually be used in several controlling functions.

In this paper we introduce a concept to integrate all components of a robot environment into one programming interface to allow high level language programming and re-use of sensor control software. All devices are encapsulated to form an *execution layer* as the lowest level of hierarchy for programming multiple robots and other devices of the workcell. It is intended to simplify programming and to be used by higher level planning systems.

Section 2 examines the requirements that have to be met by an execution layer and introduces the concept of local autonomy. In section 3, the *basic services* that are made available through the execution layer are developed. Section 4 shows our implemented execution layer – the *ELWMS* – and its programming. Conclusions and an outlook end the paper.

2 Local Autonomy

The execution layer is intended to integrate all the hardware devices and the elementary software such as drivers and low-level controllers into one entity that is interfaced from a higher level system, see Fig. 1. This higher level system may be a sophisticated task-level planning system or a user-written robot application. To serve this purpose, the following demands have to be fulfilled:

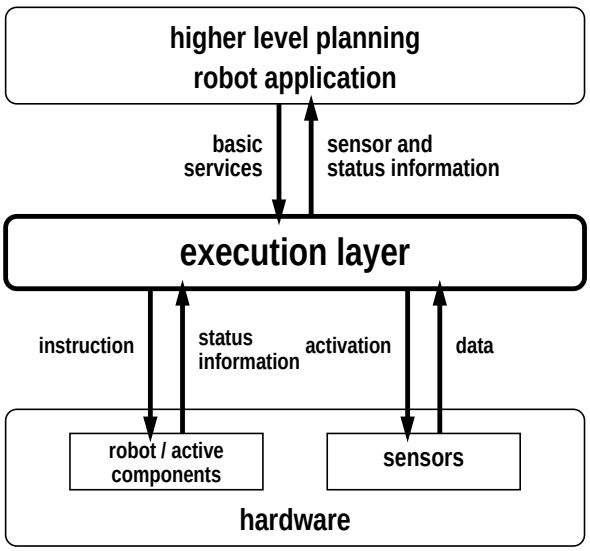

Figure 1: The execution layer connects hardware and higher level software. It is interfaced in a uniform and abstract fashion from *above* and handles all control aspects of the layer *below*, the actors and sensors.

Small Interface — The execution layer integrates sensors and actors and allows the higher level to abstract from physical details. The interface has to be independent from the technical features of the hardware components, as there will be changing hardware devices and changing tasks. Programming shall be as easy as possible. Thus the interface has to be small and adaptive to all kinds of components that are connected to the execution layer.

Handle Sensor and Status Information — To execute a task autonomously, complex actions have to be planed, based on the results of sensor inputs. E.g. identifying an object, estimating its location and orientation with image processing and range sensing, and then grip it and place it in an assembly. These kinds of sequences requires higher planing on a symbolic level and should not be integrated into the execution layer. To enable it, sensor readings and preprocessed sensor data must be made available within the execution layer to be read by the higher level system. In addition to this, hardware status information must be made available as well to give the higher level system a precise description of the current situation that allows the necessary update of the environment modelling for planning.

Sensor readings are as well required in cases of errors. The execution layer can only react in a limited fashion on exceptions and errors while executing activities as it has a limited knowledge of the environment and the task that is currently processed. A complex reaction and replaning is only possible for the higher level planning system, and sensor readings and status information about the hardware devices are needed. The necessary knowledge to replan in cases of errors has to be modeled within the higher level system.

There are cases, of course, where the action is coupled more directly to the sensor data and the processing may have to take place within the execution layer. The possible sensor-actor couplings can be classified into three major classes of actions [Hagg, 1992, Brunner, *et al.*, 1994]:

- *Sensor–Controlled Actions* — Sensor–controlled actions use the sensor values to calculate modifications of the action continuously. The sensor values are fed back into the robot motion directly, for example to compensate forces that occure due to uncertain knowledge of a gripping location. These control loops have to be a part of the execution layer.
- *Sensor–Terminated Actions* — These actions are characterized as follows: some sensor values are read continuously and evaluated to recognize either the successful termination of the action or an error situation. An example for this kind of action is the supervision of a minimum distance between the robot and a table with a laser range sensor – if the distance fells below, the action is stopped with an error. These kinds of actions have to be handeled autonomously through the execution layer.
- *Sensor–Planned Actions* — The main characteristic of these kinds of actions is the temporal decoupling of the sensor reading and the execution of the dependent activity. A typical ex-

ample is to grip an object after its localization. There are no temporal bounds, so it does not make sense to integrate these actions into the execution layer.

Real-Time Operation — Several hardware components and low level software modules have to be operated under real-time conditions. If multiple sensors are used, the point in time when data becomes available has to be managed to enable sensor data fusion. In the case of sensor supervision the reaction on passed thresholds has to be immediate. A warning or an alarm must be handled at once (see below). All sensor-actor control loops within the execution layer have to operate with the control frequency of the moving device to allow the smoothest possible operation, e.g. if the robots compensate forces while interchanging parts, this should happen without fluttering.

Error Handling — Strong real-time bounds exist as well for the time to handle errors and exceptions while running an application. At least an immediate first reaction has to be assured to stop active components and bring them into safe positions. This is required to avoid or minimize damages to the robots, the tools and the manipulated objects. Relevant error information has to be propagated to all concerned modules that are connected with the erreanous part. These modules are on the one hand lower level devices connected with the erreanous part, on the other hand the higher level system that initialized the current operation. If the robot collides, sub-devices like tools have to be stopped and all kinds of sensor and status information has to be send to the higher level planing system.

Local Autonomy — The requirements for the execution layer can be summarized under the idea of *local autonomy*. The execution layer is ordered to do *basic services*, they are only initialized by the higher level system and processed as independent as possible by the execution layer.

All real-time sensor-actor control loops have to be included in basic services. An example is the sensor-controlled sliding along an obstacle's contour. From the perspective of the higher level systems this action is only started and a terminating condition is described. The execution layer handles the sensor readings and modifies the movement accordingly without further communication with the higher level system. Error situations and exceptions as well as the correct termination have to be supervised and processed autonomously by the execution layer. The higher level planning system is then informed and has to decide how to proceed.

Local autonomy means that all activities and decisions that are possible within the execution layer are handled and made there to capsule the hardware and the lowest level of software as much as possible. On the other hand, local autonomy means to keep the responsibility and knowledge of the exection layer *local*, both in time and in relation to the task to be processed.

3 Basic Services of the Execution Layer

The execution layer has to have an application-independent kernel or basic system that offers functionality that is needed in any program. This kernel has to include such basic aspects like the motions of the robot or elementary sensor-actor control loops, e.g. to slide along obstacles or to grip fixed parts while compensating forces. With such a powerful basic system, the development of an application is reduced to arrange and parameterize the required operations. More complex applications will require the creation and integration of the additionally needed components into the basic system.

To fulfil the demands described above, the execution layer offers basic services that can be started from the higher level system. These basic services are defined as operations that are not further devided by the higher level planning. They are the primitives of all possible operations of the robot system. In general, they are initialized by the higher level system and processed autonomously by the execution layer, where they are transformed into a sequence of instructions for the different components [Hörmann and Rembold, 1991]. A closer look at the basic services allows to refine the execution layer and to focus on further aspects that are required.

3.1 Adaptability

To create an execution layer as flexible and as powerful as possible, two levels of adaptibility have to be possible:

Adaptability of the Basic Services — To serve for all purposes, the basic services shall be as independent as possible from the parts to be manipulated, the manipulation sequences and the environment. Thus it is necessary to make them adaptable to all kinds of possible tasks. This can be done by passing all required information as parameters when the basic service is called or by configuring the basic service with a seperate call to get a task-specific behaviour. An example would be to select the axes to be controlled by a general force-torque controller module. This allows to develop very advanced, general and adaptive controllers that can easily be reused in other applications.

Adaptability of the Execution Layer — Even with very advanced adaptive basic services it is in general impossible to reflect all possible requirements for all kinds of tasks. Thus the execution layer has to be adaptive itself, allowing to develop and integrate new basic services. This adaption does not have to be possible on-line and will in general require to recompile and relink parts of the kernel system. But these extensions should be part of the concept of the execution layer and should not endanger the integrity of the existing system. New basic services become necessary when, for example, a device should be controlled by a sensor that was not used to control this device before, or – of course – when a new sensor or actor is installed in the system.

3.2 Scope of the Basic Services

The basic services of the execution layer have to cover the following classes of activities:

Control of Hardware Devices — To enable the higher level planning system to use active components, basic services to control hardware devices directly are needed. Examples for these basic services are movement commands for the robots, opening and closing commands for the grippers etc.

Control of Cooperating Manipulators – To enable the cooperation of two (or more) robots, basic services are required that allow to command cooperative manipulation directly. Two different classes of cooperative manipulation can be distinguished, that have to be reflected in the basic services:

- *Point Coordination* — The first class of cooperation refers to two or more robots with overlapping workspaces, each working on independent tasks. The only continuous coordination is to assure collision free movements for the robots. The motions are only synchronized at certain points. If a part is interchanged between two robots, the motion is synchronized at the interchange position only.
- *Trajectory Coordination* — This second class of cooparation is given when there are fixed dependencies between the tool coordinate systems of the cooperating robots, i.e. the robots manipulate one object together. This requires a close coupling of the motion of all participating robots. The trajectories have to be synchronized continuously. An example is the transportation of a part with two manipulators. This results in a closed kinematic chain, and the necessity of trajectory coordination is obvious to avoid damages to the robots and/or the manipulated object.

Activation of Sensor Systems — Basic services are needed to activate and control the sensors. This does not only mean access to the sensor hardware, but to operate the sensor system as a whole, e.g. being able to initialize and calibrate a sensor, switch its modes etc.

External Feed-back of Sensor Readings and Status Information — The higher level system has to be able te request any kind of data that is available within the execution layer to be able to plan and to react.

Internal Feed-back of Sensor Readings — Another important task for basic services is the transformation of sensor inputs into actions of active components. It is not sufficient to enable the control of hardware devices and sensor systems alone, but sensor values have to influence active components directly.

4 Implementation

We implemented an execution layer for our laboratory system, according to the concept developed above. We use several sensor systems (force-torque-sensor, laser based range finding sensor, LED-array) on two Unimation industrial robots. Both robots are equipped with servo grippers and are mounted so close to each other, that their workspaces overlap and cooperative manipulation is possible.

The execution layer is realized as a client-server system. The configuration and the communication are shown schematically in Figure 2. The system is called *ELWMS*, an abbreviation for **E**xecution **L**ayer **W**ith **M**onitored **S**ecurity.

4.1 Configuration and Communication

The client can be the upper planning system or any other user written robot application program. To use the *ELWMS*, it has to run under UNIX operating system and has to have a connection to the Internet. The required code is made available as a C++ class library. The client contacts the servers through remote procedure calls.

The two servers are connected to the sensors and actors and run under the real-time operating system *LynxOS*. All basic services are executed through the servers. The basic services are initialized only, the client does not wait for the termination. Information on the status of execution of basic services and other hardware information is returned to the client asynchronously via callbacks.

Each server is connected with one of the robot control units and the hardware connected to the respective robot, like its gripper and the integrated sensors. To synchronize the two robots, two dif-

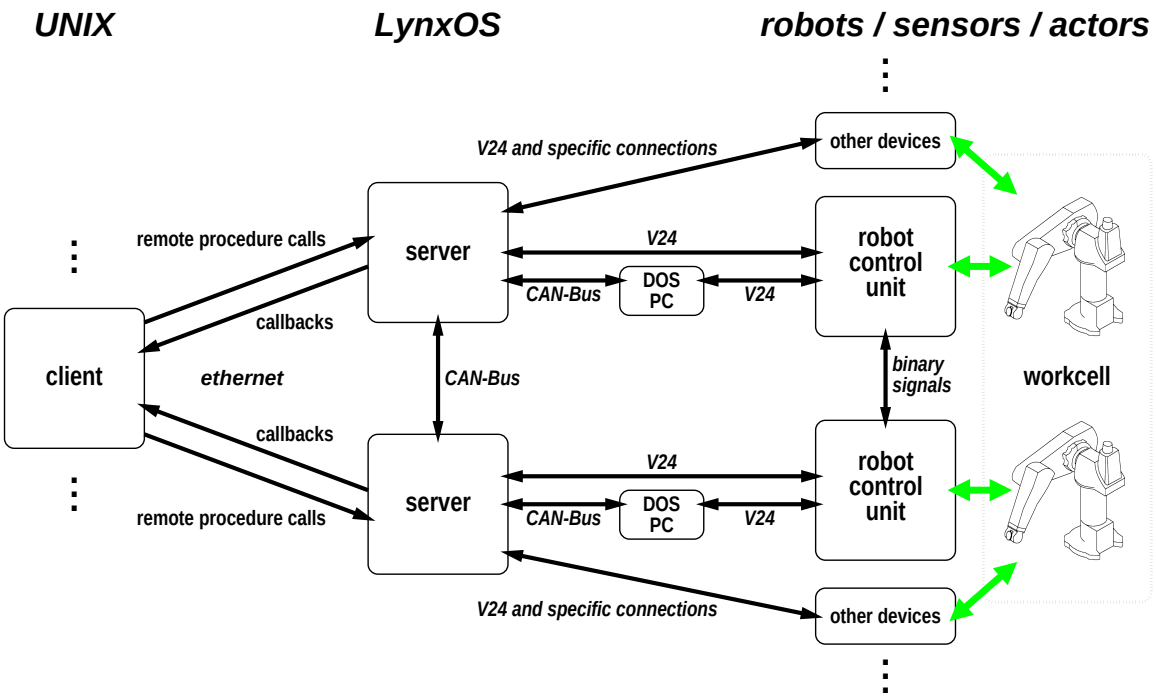

Figure 2: The main communication structure of the *ELWMS*. User programs or higher planning systems run under UNIX and are connected via Ethernet to two servers operating under LynxOS. These real-time servers are connected via a bunch of V24 and other specific connections to all hardware components of the robotic workcell.

ferent systems are provided. Between the robot control units binary signals are used to enable *point coordination*. To synchronize whole trajectories, the servers are connected via CAN-Bus, an efficient industrial field bus.

The two DOS-PCs in the middle of Figure 2 are used as intelligent V24 interfaces that assure to serve the ALTER–interfaces of the robot control units in the rhythm of the joint controllers. This is required for any on–line movement modification, thus the PCs are an important link within all closed real-time control loops.

4.2 Basic Services and Programming

The basic services of the *ELWMS* are distinguished in three classes:

Elementary Basic Services — The elementary basic services represent the fixed hardware components, and form a complete set of functions to drive them. An example for this class of basic functions are the standard movement commands for the robots, as well as the commands for synchronized trajectories.

Adaptive Controllers — To represent sensor-actor mappings and to enable any kind of closed real-time sensor control loop, adaptive controllers are used. An example for this class is a force-torque controller that modifies the robot's motion according to a parameterized strategy, e.g. to follow a two dimensional contour or to be moved in all six degrees of freedom by moving the tool manually. A LynxOS–thread is created for each controller started by the higher level system. The function is called in the rhythm of the robot joint control unit ($28ms$). This results in a quasi–parallel processing of the seperate controllers. Each controller function returns values to correct the motion of the robot arm. Within the execution layer, all these values are accumulated and send to the robot joint control unit.

Adaptive Task Monitors — These are used to enable any kind of supervision while the operation is going on. An example for this class is a task monitor that keeps an eye on the robots tool position in cartesian space. If it leaves a predefined region the robot is stopped. The adaptive task monitors are started as a seperate thread and called in the rythm of the robot joint control unit. The only difference to the adaptive controllers is that the task monitors do not calculate data to modify the robot's motions. They are as well operated (quasi–) parallel in individual LyxOS–threads.

The adaptive controllers and the adaptive task monitors allow to adapt the ELWMS to new requirements, either new sensors or actors, or new algorithmic concepts for sensor-actor couplings.

New adaptive controllers and the adaptive task monitors have to be programmed by the user and can be integrated into the execution layer in the simplest possible way. They have to follow a convention for the parameterization and are listed within the execution layer by name. Any parameters and further information can be passed to and returned from the controllers and the task monitors in a string of arbitrary length and format. To extend the execution layer, the C++ class library under UNIX does not have to be modified, only the server program has to be rebuilt.

The adaptive controllers and task monitors are not application specific and can be reused to configure a server as demanded. They form a library of functions for sensor and status data processing and integration. By now, we have developed several very advanced adaptive controllers, e.g. one for the force-torque-sensor that allows to use any combination of measuring directions to control any degree of freedom of the robot's tool. The controller knows several modes of compensation to assure correct operation even with uncentered load. If, for example, all degrees of freedom are read in and controlled, the robot's tool can be moved manually by the user [Hockauf, *et al.*, 1995].

The higher level system, on the one hand, has to have enough knowledge about the whole environment to ensure an appropriate sequence of calls to basic services, e.g. not to use a sensor before it is initialized. The execution layer, on the other hand, has to do all the scheduling of the basic services an run them simultanously or in sequence as necessary. By concept, the number of active controllers and task monitors is not limited and the only practical limit is the computational power of our real-time server computers.

5 Conclusion

A concept for a powerful execution layer has been developed. Our implementation shows that the main goals can be reached. It has become possible to program the robots and all their accompanying subsystems through one interface, integrated in one library. The programming of applications has turned into a sequence of basic service calls, much more convinient then the use of several libraries and/or communication protocols for all the devices in use.

An execution layer that capsules the underlying hardware systems opens a way to develop more advanced robotic control software as superior layers. These layers will be *more autonomous* then the execution layer, they will have a less local autonomy and will be responsible for more complex tasks and their processing.

Bibliography

[Brunner, *et al.*, 1994] B. Brunner, K. Arbter and G. Hirzinger (1994). Task Directed Programming of Sensor Based Robots. In: *1994 IEEE/RSI/GI International Conference on Intelligent Robots and Systems*, pages 1080–1087, Munich, Germany.

[Hagg, 1992] E. Hagg (1992). *Realisierung von Multisensoranwendungen mit vernetzten Logischen Sensoren und Aktoren.* Ph.D. Thesis, Technische Universität München, Verlag Shaker, Aachen, Germany.

[Hockauf, *et al.*, 1995] M. Hockauf, R. Lechner and M. Müller (1995). *Sensorkonrollierter Austausch eines Objekts zwischen zwei kooperierenden Robotern* Fortgeschrittenenpraktikum, Technische Universität München, Germany.

[Hörmann and Rembold, 1991] A. Hörmann and U. Rembold (1991). Development of an Advanced Robot for Autonomous Assembly. In: *1991 IEEE International Conference on Robotics and Automation*, pages 2452–2457, Sacramento, California, USA.