Proceedings of the 2000 IEEE
International Conference on Robotics & Automation
San Francisco, CA • April 2000

BERRA: A Research Architecture for Service Robots

Mattias Lindström
mattiasl@nada.kth.se

Anders Orebäck
oreback@nada.kth.se

Henrik I. Christensen
hic@nada.kth.se

Centre for Autonomous Systems,
Royal Institute of Technology,
Stockholm SE-100 44, SWEDEN.

Abstract

The architecture for a mobile service robot is discussed. The paper covers aspects related to overall design and implementation. The robot architecture is of the hybrid deliberative/reactive behavioral type. The strategy is selection where planning is viewed as configuration. The architecture can be divided into three layers, one for deliberation, one for task execution, and one reactive layer. Scalability and a high degree of flexibility have been primary design goals, making it easy to reconfigure the system. The system has been built in an object oriented fashion. An application of the architecture has been tested in a significant number of missions in our lab, where one room has been setup as an ordinary living room.

1 Introduction

A fundamental basis of any intelligent system is the underlying architecture, which provides the necessary coordination, communication, and control structures. In this paper we present BERRA (BEhavior-based Robot Research Architecture), a research architecture used in the development of a service robot. The objective of the service robot is to carry out a wide range of missions in an ordinary home or office setting. The missions can for example be fetch-and-carry operations, such as delivery of mail or give tours of the office to visitors.

1.1 Service Robot Architectures

A robotic system used in an ordinary household must be especially robust and reliable and must be prepared to handle unexpected situations. The architecture of the system must provide a framework for robust integration of required skills. For research and development purposes, an architecture should provide support for:

- A conceptual framework for reusability
- A clear distinction between levels of competence
- Simple integration of new components
- Flexibility
- Efficient run-time performance
- Simple debugging

If the architecture is going to be distributed to other research labs, or used on a variety of OS, the code should also be written in a standardized language and OS specific system calls should be wrapped.

2 Analysis of Architectural Requirements

Before the architecture is outlined, an analysis is conducted that establishes requirements and demands. It will focus and enlighten issues for the design.

The design should provide a conceptual framework for research on integration of applications and skills on a service robot. A service robot requires deliberation in the form of autonomous planning and communication with humans. At the same time a service robot must be able to react instantly to unexpected situations. Neither deliberation nor reactiveness can be ignored in favor for the other. Therefore we choose a hybrid deliberation/reactive behavior-based approach.

To accommodate the demand of flexibility, task selection is carried out according to the *planning as configuration*[1] strategy. That is, modules in the reactive layer can be configured and connected together in a flexible network.

The initial hardware which the architecture is applied to, is our Nomad 200 platform (called Asterix). Equipped with numerous sensors, such as sonars, IR-ranging, laser range scanner, pan/tilt mounted b/w stereo camera pair, and a color camera, it embodies a reasonable complex platform for the testing. The architecture must also scale to our Nomad XR4000 robot (called Obelix), equipped with three onboard computers, and to our multiple robot team of Nomad Scouts (called Hewey, Dewey, and Lewey). The architecture can therefore not be limited to run on one computer, but must be able to span over a network of computers.

2.1 Deliberate layer

The deliberate part of the architecture needs to understand commands given by a human operator. To fit in to its the service role, it would be convenient if it was able to communicate in the same manner as humans. This calls for the need to understand and express itself using speech and gestures.

We do not only wish the robot to understand the operator, but also to effectuate the command in a satisfactory manner. This requires the need for reasoning and planning of missions.

A mission often involves transportation. The robot needs to know where it's located and understand names of locations (destinations) given by the human operator. To travel to a new location in an effective way, the robot needs to perform path planning. The path planning should also exploit the possibility to learn by experience and become more effective.

There is no guarantee that a mission will complete successfully. The deliberate layer must be able to intercept and recover from a failing mission.

2.2 Reactive Layer

In the reactive layer of a behavior-based architecture, behaviors act as tight couplings between sensors and actuators. To reduce complexity, each behavior should have a simple and well defined task that depends on as few other components as possible. By minimizing the complexity of each behavior the architecture can allow a large number of behaviors. The architecture then scales better to more multi-faceted situations.

Sensors will be used simultaneously by different behaviors. Thus, a mechanism for sharing the sensors has to exist. Some sensors take quite some time to read. To make the system more effective, the reading of the sensor data could be done once for all concerned behaviors. Also some basic refinement could be done in common.

Similarly the output to the same actuator from different behaviors could result in an undesirable and erratic behavior. This means that there has to be some sort of fusing mechanism involved.

Since this layer controls the motion of the robot, a near real-time performance is necessary. The sensor data must not be allowed to get too old before they effect the actuators.

This layer will grow as new functionality is needed, thus a generic framework will reduce development time.

2.3 Task Execution Layer

There is a need for components that bridge the gap between the deliberate and the reactive layers. The reactive layer network needs to be configured and monitored according to the deliberate layers decision. There is also a need for a functionality that administrate all components and their whereabouts.

3 The BERRA Architecture

We have chosen to have the components as individual processes, since concept of a behavior-based architecture maps nicely onto processes and sockets.

To ensure compatibility, the standardized language C++, is used. Also, together with the ACE[2] package, portability over a wide range of OS is maintained.

The architecture is designed so that components can be transparently placed on any machine in the network, which infers that the system can be run on multiple computers.

The system is throughout constructed using an object-oriented approach. Abstract base classes have been implemented to aid in the making of new modules. Creating for example a new behavior is done rapidly and assumes only basic knowledge of the architecture.

3.1 Deliberate Components

Human Robot Interface The human-robot interface (HRI) is the link between the human and the robot. In our system the HRI understands both gestures and speech. A voice synthesizer is used for audible feedback.

To give the robot and the human a common reference of locations, a concept of goal-points is used. A goal-point is a useful location in the world that has been given an intuitive name, for instance *dinner-table* or *mailbox*.

Planner The main objectives of the planner is to interpret and fulfill the commands given from one or more human operators though the HRI. By using prior knowledge, such as a topological database, the planner will convert the command into a list of consecutive states and state data. Each state represents a certain configuration of the reactive layer. The state data typically represents a goal-point given in global coordinates. These states and data will be sent one at the time to the layer below (called task execution layer). Upon receiving the message of success, the next item on the list is conveyed. In the case of receiving the message of failure, the plan is revised. If the planner cannot recover from the failure, the human operator is informed.

3.2 Task Execution Layer

The Task Execution Supervisor The *Task Execution Supervisor* (TES) has the function of managing the reactive control layer. It receives a state change from the planner and translates that into a configuration of reactive components. According to the state, TES informs the controllers what behaviors they should receive data from.

It also sends relevant state data to the behaviors. At startup, TES reads a configuration file which contain necessary information about the behaviors, and which behaviors that are to be associated with each state. This file can be re-read during execution, to allow for online re-configuration.

Localizer The *localizer* holds long term information and is therefore not placed in the reactive layer. Its function is to find out where the robot is located in the world, and then track this position. When necessary, relocalization will be performed. Other parts of the system can connect and subscribe to this information.

It also translates global (world) coordinates into local (odometric) coordinates. This is needed by behaviors depending on goal-points. A behavior depending on goal-points will automatically receive any correction of its position from the localizer.

3.3 Generic Reactive Components

Resource To facilitate the sharing of sensor data, we have created a base class named *resource*. A resource is a server, where the typical clients are behaviors (see below). However, a resource can also be a client to another resource, in order to produce higher level abstraction of data. When a resource has no clients, it automatically enters an idle mode, thereby reducing the load on the computer.

A client who wants to receive information from a resource must first establish a connection and request a subscription scheme.

Presently, three subscription types have been implemented:

- The client send requests each time it demands new data (the pull paradigm).
- The client will be updated each time new data is available from the sensor (the push paradigm).
- The client specifies a time interval between updates (the synchronous paradigm).

There are a number of aspects to consider when designing subscription schemes. One is that it's advantageous in cost to send few large chunks instead of many small. Another is to reduce the amount of transmitted data by sending a reference to a shared memory location instead of the data itself.

Behavior The tight coupling between sensors and actuators are realized in *behaviors*. A behavior can connect to one or more resources and allow connections to made by a controller (see below). The production of actuator data can be either time- or data-triggered. The latter case refers to when a new set of sensor data is pushed from a resource. The behavior can accept configuration data from TES. It will also, if relevant, inform TES of success or failure. When a behavior is data-triggered, a timeout period can be set that states how long time it can wait between sensor updates. If it expires, a control value, resulting in the stopping of the actuator, will be produced.

Controller *Controllers* are responsible for sending direct commands to the robot's actuators. Upon a change of state, the controller receives new directives from TES on what behaviors the controller should get data from. The controller then connects to these behaviors. The data received from the behaviors is fused or arbitrated to produce a control signal to an actuator. This behavior fusion mechanism has to be specified in the implementation of the controller. It could

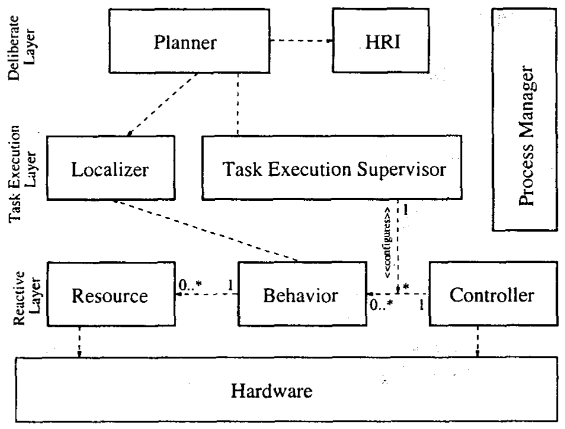

Figure 1: A model of the implemented architecture

for example be an arbitration or a weighted vector summation of the proposals from the behaviors. Other schemes use fuzzy logic or voting.

If a specified timeout period between behavior updates expires, the actuator will be stopped. Also, upon receiving a stop command from TES, it stops the robot and disconnects the behaviors and enters an idle mode.

3.4 Process Manager

The *process manager* (PM) keeps track of all processes used in the system. It maintains an archive for this information, where each process has to register its name, host, and address[1]. The name serves as a unique identification key for the process and is used as a reference by other processes in the system. If necessary, the PM will also start the requested system component. The functionality is similar to the CORBA Nameserver[3].

PM has one or more executor daemons, one on each computer involved. On demand from PM, an executor will start and kill processes used in the system. The executors can be viewed as servants of PM, that gives possibility to reach over all computers used in the systems. The executor will only act upon requests from PM, except when the communication to PM is broken. In such an event, the system is inevitable going down and the executor will kill all the processes it has started and return to an initial waiting state. The executor daemon is started once at the boot of the computer.

An overview of BERRA is shown in Fig. 1.

[1]port number or file descriptor

3.5 Communication Issues

At initialization of a process, a port number chosen by randomization, is registered at PM. By asking PM, any other process can set up communication with the process. The advantage of this scheme is that ports are not hard-coded and will not conflict with each other. The port number to PM is, however, given to all other processes through the executor on each computer in the system.

Each process in the system will have technically two entry ports. One local for clients on the same computer (using UNIX socket communication) and one global for clients on other computers (using INET socket communication). Judging from the address to the server the client may, if on the same computer, establish communication with the faster local port.

3.6 Timing and Data flow Issues

For reasons of performance and scalability, issues of time and data flow will be briefly discussed. Synchronization, and how the data flow is organized, has a great impact on a systems performance.

In order to address these issues, the following steps have been taken wherever possible:

- Processes are data- instead of time-triggered
- Data is being pushed instead pulled
- Processes are started on a when-needed basis
- Processes enter an idle state when not used
- Memory consuming objects are only instantiated in the active time period

In the reactive layer, timing is of particular importance. Though, real-time in a strict sense is not called for, a high and stable quality of service is needed. For obvious reasons, the most important behavior loop is the one including *avoid obstacle.*

The time measured from the moment the sonar resource has collected sonar data, to the moment this data has contributed to a motor command, was in the early version of the system measured to several hundreds of milliseconds with a great deal of variance. After switching to the above outlined scheme, the time is reduced to 14 milliseconds with very little jitter.

4 Integrated Applications

The BERRA architecture has been tested[4] with the following reactive components: *avoid obstacle, door* *traverse, go point, explore, visual follow, visual find, laser cornerdocking, mail docking, vehicle controller, sonar resource, IR resource, laser resource and image resource.* In order to evaluate the robot system, one of our rooms in the laboratory has been transformed into a furnished living room. Navigation within our office and living room is very robust. The system has been tested in hundreds of missions and can run for several hours without failures. The operator can command the robot to go to a number of predefined goal-points. For example, the spoken command, *Robot, go to the dinner-table*, will make the robot drive to the dinner-table in the living room. Other more complicated tasks can also be performed. For example perform a guided tour where the robot will present different rooms in the lab.

4.1 Example Mission

An example of the mail delivery mission is described in detail. The robot is switched on in a room without any prior knowledge except for a map of collected landmarks. The operator orders the robot to find its current position. The planner, receiving the order, sends the state command *explore* to TES. TES finds in its list that the two behaviors *explore* and *avoid obstacle* are associated with this state. These two names are sent to the vehicle controller who requests their locations from PM. Since these processes do not exist yet, the executor starts them. The behaviors will, in the same manner, invoke necessary resources. Now, their resources can push sensor data to the behaviors, who will in the same manner push steering proposals to the vehicle controller. At the same time, the planner asks the localizer to localize globally. Once the localizer finds its location, the planner sends *stop* to TES, which is relayed to the vehicle controller who issues halt to the motors and disconnects the behaviors. Since the behaviors are no longer used, they disconnect the resources and become idle. Then, the robot is commanded *deliver mail in the living room!* The planner plans a route from the current position to the mailbox in the corridor. The plan is stored in a sequence of tasks, which is sent to TES one at the time. The behaviors *go point* and *avoid obstacle*, takes the robot to a position just before the door to the corridor. When the *go point* is done, the planner is notified and sends the state *door*. After the behavior door traverse has brought the robot safely through the door, go point and avoid obstacle takes it to a position close to the mailbox in the corridor. For the final positioning, a laserscanner behavior is used to align the robot to a corner. This has to be very accurate, since the actual mail pickup is done by dead reckoning. After the picking up the mail, the robot returns and announces the new mail.

4.2 Related Work

Arkin was one of the first to advocate the use of a hybrid control architecture[5]. The approach was implemented in the AuRA architecture, which is a rather specific system for experimentation with behavior-based systems. The system was specifically designed for operation on a single platform that carries out navigational tasks.

The ROMAN system[6] was designed at TU Munich for health care services and domestic automation. This system is designed around a common blackboard architecture to which a set of 'expert' modules are connected. Examples of experts are obstacle detection, locomotion, and planning. The system has been successfully evaluated in a range of different settings, using specific models for the task domain (i.e. CAD models of buildings).

The XAVIER system[7], developed at CMU, is composed of four layers: task planning, path planning, navigation, and reactive behaviors. The system is built from a set of standard components for communication, planning, and behavior integration. The system has been in almost daily use since December 1995. It has served almost 2500 navigation requests from the Internet with a completion rate of about 95%. The RHINO[8] system developed at Bonn University is similar to the XAVIER system and is used as a tour guide in museums.

Common to all of the above system and many others reported in the literature is that they exploit a well defined architecture, where the different components are tightly integrated. This benefits performance at the cost of generality and reusability.

At Vanderbilt university, a dual-armed stationary humanoid robot, ISAC, is developed. It's meant to be uses as robotic aid system for the service sector such as hospitals and homes. The architecture is called IMA[9] and has some similarities to BERRA. It's a two-leveled framework, the robot/environment level and the agent/object level. The former, is a group of coupled computing modules, similar to our reactive layer. The latter, consists of a group of components that describes configurations of modules, similar to the states in TES. An important difference compared to BERRA is that deliberation is mixed into the modules and that there is no demand of tight coupling to actuators.

The 3T Architecture[10], and similarly AAA[11], consists of three layers, controller, sequencer, and deliberator, that group algorithms into, respectively, those with real-time, fast, and slow responses. The architecture is similar to BERRA in many ways, but differs in for example that 3T has a centralized sequencer, while BERRA has true concurrency between all the behaviors and modules in the system. This also gives the strength to distribute the load of the system over multiple computers at module granularity. In 3T there is no distinction between a skill that combines information into an actual motor command and other skills. In BERRA there is a special framework, controller, to fuse outputs from behaviors into motor commands.

5 Conclusions and Future Work

This paper presents a hybrid deliberate/reactive architecture for a service robot. One key goal in the design has been to make new functionality easy to integrate. Researchers who wish to perform experiments can for example, with little effort and knowledge of the underlying system, construct a new behavior. The architecture is scalable, to allow for the competence of the robot to grow. New devices are similarly simple to add. This is accomplished using an object-oriented approach.

The immediate future work consists of interfacing the system architecture with the manipulator control on Obelix running QNX. The proposed solution[12] is to consider the manipulator as a behavior.

Acknowledgments

This research has been sponsored by the Swedish Foundation for Strategic Research through the Centre for Autonomous Systems.

References

[1] R. C. Arkin. *Behavior-Based Robotics*. The MIT Press, Cambridge, Massachusetts, London, England, 1998.

[2] D. C. Schmidt. The adaptive communication environment: Object-oriented network programming components for developing client/server applications. In *In Proceedings of 11th and 12th Sun Users Group Conference*, 1994.

[3] M. Henning and S. Vinoski. *Advanced CORBA Programming with C++*, chapter 18, pages 771–772. Addison Wesley, 1999.

[4] M. Andersson, A. Orebäck, M. Lindström, and H. I. Christensen. ISR: An intelligent service robot. In *Sensor Based Intelligent Robots*, volume 1724 of *Lecture Notes in Artificial Intelligence*, pages 287–310. Springer Verlag, 1999.

[5] R. C. Arkin. Towards cosmopolitan robots: Intelligent navigation in extended man-made environments. Technical Report COINS 87-80, Ph.D. Dissertation, Dep. of Computer and Information Science, 1987.

[6] U. D. Hanebeck, C. Fischer, and G. Schmidt. Roman: A mobile robotic assistant for indoor service applications. In *Proceedings of the International Conference on Intelligent Robots and Systems*, pages 518–525, 1997.

[7] R. G. Simmons. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation*, 10(1):34–43, 1994.

[8] M. Beetz, W. Burgard, A. B. Cremers, and D. Fox. Active localization for service robot applications. In *Proceedings of the International Conference on Intelligent Robots and Systems*, pages 175–186, 1997.

[9] R. Pack, M. Wilkes, and K. Kawamura. A software architecture for intergrated robot development. In *IEEE Conference on Systems, Man, and Cybernetics*, pages 3774–3779, Orlando, FL, September 1997.

[10] R. P. Bonasso and D. Kortenkamp. Using a layered control architecture to alleviate planning with incomplete information. In *AAAI Spring Symposium on Planning with Incomplete Information for Robot Problems*, March 1996.

[11] J. R. Firby and M. Slack. Task execution: Interfacing to reactive skill newtworks. In *AAAI Spring Symposium on lessons learned from Implemented Architectures for Physical Agents*, Stanford, Calif., March 1995.

[12] L. Petersson, M. Egerstedt, and H. Christensen. A hybrid control architecture for mobile manipulation. In *Proceedings of the International Conference on Intelligent Robots and Systems*, pages 1285–1291, Korea, 1999.