Components in Intelligent Robotics

Anders Orebäck
Center for Autonomous Systems
Royal Institute of Technology, SE-100 44 STOCKHOLM, Sweden
oreback@nada.kth.se

Abstract: Research on mobile intelligent robotics has been carried out for some decades now, with major improvements made during the last fifteen years. However, very few applications have been commercialized and moved outside the research laboratories. Several researchers in the robotics community believe that one important reason for this is the fact that very little of the work carried out can be shared between labs and cross evaluated. This is a consequence of a lack of a common framework, as well as writing hardware independent software. In order to address this problem, work has been initiated at several places to introduce the concept of components in the field of robotics. The aim of this paper is to survey this work, and to discuss the associated problems.

1 Introduction

The building of a complete mobile robot system, requires expertise in a number of different disciplines such as automatic control, computer science, sensor knowledge, mechatronics, artificial intelligence, etc. In order to make significant progress, most robot researchers are specialists in one of these areas. Nevertheless, a complete integrated system is needed to validate any small piece of the puzzle.

There are only a few labs in the world that have the resources needed to build a complete robot system from scratch. Even though most researchers now adhere to a similar architectural approach, compatibility between different software systems is non existing. Instead, researchers develop for a specific software system with no generalization in mind. Moving parts from one system to another is thus a big effort. This prevents sharing as well as comparison of parts with the same functionality, but based on different algorithms.

The software engineering community have been addressing the issues of reuse and components for several years. It is high time for application fields like robotics, to adopt the knowledge, methods, and tools created.

A component based solution would simplify the following:

- exchange of software parts between labs, allowing specialists to focus on their particular field.
- comparison of different solutions.
- startup in robot research.

We and others believe that the introduction of components is a necessary step to move the robot technology from the research labs to commercial business applications.

This paper starts with a look on how robot system have been built in the past. We then look at the available component technologies. Next, a survey is made on component based work within robotics. We end the paper with conclusions. The paper focuses on the branch of service robotics, although most herein can be generalized to other areas of robotics.
2 Robot Architecture

This section provides the requirements of a robot architecture and the implications these give to a component designer. Some background information is also given for the readers unacquainted to the field of robotics.

2.1 Robot architectural requirements

A fundamental basis of any intelligent system is the underlying architecture. 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 to give tours of the office to visitors.

A robotic system that should be 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 the 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 and devices
- Flexibility
- Efficient run-time performance
- Simple debugging

2.2 The hybrid deliberate/reactive robot architecture

A tremendous amount of research has been put into robotics in the last fifteen years. Traditionally the architecture of the software-systems were of a hierarchical type, highly influenced by the AI research of the time. This meant a system having an elaborate model of the world, using sensors to update this model, and to draw conclusions based on the updated model. This is often called the sense-plan-act paradigm.

The systems did not perform very well, partly because of the difficulty in the modeling of the world, partly because of relying too much on inadequate sensors. In 1987 Rodney Brooks revolutionized the field by presenting an architecture based on purely reactive behaviors with little or no knowledge of the world, see e.g. [7]. A hybrid approach has since then been common among researchers e.g. [2].

The reactive subsystem of a hybrid system is often behavior based. This means that the subsystem consists of separate behaviors, where each behavior has one specified non-complex task. Examples of behaviors are follow-wall, traverse-door, and avoid-obstacles. These behaviors represents a tight coupling from the sensors to the actuators. Since several behaviors can be active at the same time, the results must be fused into a single crisp actuator command. This is done in a controller module. The reactive layer only handles short-term information. A deliberate layer is required to handle tasks which needs higher level abstractions such as mission planning, localization, and interactions with an operator. A third layer, often called either the configuration layer, or task execution layer, is usually needed to bridge the gap between the deliberate and the reactive layers.

A can be seen, the hybrid deliberate/reactive behavior based architecture already has clear distinctions of competences and can easily be divided into components.
2.3 Some successful robot systems

Several systems have been implemented and made to function well. The ROMAN system[13] 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. Example experts include obstacle detection, object recognition, locomotion, manipulation, planning and a human-robot interface.

The XAVIER system[22, 18] has been developed at CMU. The system 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. The RHINO[4] system developed at Bonn University is similar to the XAVIER system and has been 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.

2.4 Domain specific problems from a component view

The problem of component based robotics can be divided into finding the following:

1. Hardware abstraction.
2. Control description.
3. Interaction description.
4. System description.

To provide portable software components, it is necessary to provide an abstraction of hardware, such as sensors and actuators. The abstraction at the same time makes it more difficult to exploit special purpose sensors and hardware and there is thus a delicate balance between abstraction and efficiency that must be investigated. The control description provides a formal model of the function, i.e. perception-action, that is implemented by a particular module. It is here of interest to develop a ‘skill language’ that provides a model of the actual control carried out. One example of such a model is discrete event models.

The interaction description specifies the dependency and interaction between different software components. The system description specifies the task level coordination of modules to provide overall functionalities. It also describes methods for fusion of results from different modules. It should also provide methods for supervision of the actual execution carried out by different modules, to enable detection, recovery and handling of errors.

3 Component Issues

Although component design in robotics concern both software and hardware units, this paper is focused on software components. A large number of definitions of the term component exist. One definition is “A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to third-party composition.”[23]. A simple and quite sufficient one is that a software component is a unit of software with clearly defined interface and functionality. This section intends to identify some aspects and the current status of component usage.
3.1 Granularity

When constructing a component scheme, granularity is a major issue. At one end, we have a simple class in an object-oriented language (see section 4.6). At the other end, we can have an arbitrarily large complex system. Components can be separate processes, threaded or non-threaded in the main application. Most would agree that for practical reasons, the granularity level should be placed in the process/thread level. Moreover, reusability implies that the components should be moderate in size and functionality.

For commercial reasons, black box components should be allowed. Available source code must then not be a requirement. Although traditional object-oriented languages have the concept of interfaces, the calls to these interfaces are hidden within the private parts of the classes. This makes it difficult to replace an object within a large system. Some kind of wrapping is needed to remedy this problem.

3.2 Modeling

A number of modeling techniques have recently evolved. The most important one is the Unified Modeling Language (UML), see i.e. [6]. With UML, several aspects of a system can be modeled, such as classes, components, deployment, activities and states. By using OCL (Object Constraint Language), a subset of UML, constraint and conditions can also be modeled.

Patterns and frameworks are other methods that can be employed when constructing a system. The Catalysis methodology [8] is a very interesting approach that deals with components, frameworks and specifications in a rigorous way. Some of these techniques can probably successfully be used to implement the descriptions mentioned in section 2.4.

3.3 Component Technologies

A key component in a robot system is a reliable and efficient communication mechanism. Components need to exchange data and events. Runtime monitoring and error handling are also of great importance. Since processes may be distributed on a number of computers, host-to-host communication should be transparent. This requires a broker object that can assist other objects in finding each other. A minimum requirement is to be able to activate and deactivate a module at runtime. Dynamic reconfiguration of interconnections should also be possible at runtime. For scalability reasons, dynamic objects and process invocation on a live-when-needed basis is also highly desirable. Modern tools have factory patterns that aid in this. Data transfer can be initiated in either a pull or a push fashion.

In the past, several middle-ware standards such as sockets, Remote Procedure Calls (RPC), TCX[10] or ACE[20] were developed. Recently, a number of new interesting frameworks have emerged. These lift the problem of distributed programming to a higher level of abstraction. The most well known of these are CORBA, ActiveX, and EJB. CORBA(Common Object Request Broker Architecture) is a standard managed by the Object Management Group. Microsoft’s ActiveX is a set of technologies based on COM (Component Object Model). Enterprise Java Beans (EJB), is a specification just like CORBA. All these technologies can interact with each other and they all adhere to the Interface Description Language (IDL). They provide component models and integration frameworks, which means that we can use them for the interaction as well as for the system descriptions.
4 Ongoing Work

This chapter presents some of the efforts being made towards component usage in robotics. The author of this paper had some difficulty in finding material for this section. Moreover, the material available on some of the presented work was rather limited.

4.1 NIST Component Specifications

The American National Institute of Standards and Technology (NIST), has been developing specifications of software components for intelligent control, a domain that includes robotics. In [9], a three-stage component specification approach is proposed.

Step 1: Generic Categories for the Specifications  Initially, a set of appropriate questions are defined that are relevant to the particular class of components. These questions might include:

- What problem is this component intended to solve?
- What is the input data?
- What is the output data?
- How robust is the component?
- What kind of computing hardware is required?
- How does the component perform against available benchmarks?
- ...

Step 2: Natural Language Instantiation of the Specification  In this step, the questions from step one are answered for the particular component. The natural language description can then be made public. Search engines can locate candidate components and users can browse through the descriptions as they would through a hardware components catalog.

Step 3: Formal Language Instantiation of the Specification  At the final stage, the component is described in a formal language in order to provide an unambiguous specification and to support simulation and verification. At NIST, they used the formal language EXPRESS. They have not done an extensive survey to find the best one. EXPRESS was chosen mainly because the staff was familiar with it.

Analysis  The work carried out at NIST is of course very important. The specification part is fundamental to the component approach. The formal description in step 3 is often overlooked, because it is generally seen as difficult. Referring to 2.4, this addresses the problem of control description. The other descriptions are not considered. The question is whether defining a specification framework, without moving closer to implementation, will attract the attention of the robot community.
4.2 BERRA

BERRA (BEhavior based Robot Research Architecture) [16], is an architecture with the primary design goals of scalability and flexibility. Component thinking and reuse are dominating features. All components are heavy weight processes and can be transparently placed anywhere on the network. The implemented system makes heavy use of the ACE (Adaptive Communication Environment)[20] package. By using this package, OS dependent system calls are wrapped allowing for portability across a wide range of platforms. ACE also includes powerful patterns for client/server communication and service functions [21] which are used in the system. Fig. 1

The implementation is programmed in an object oriented fashion using C++. A standard set of base classes are used to allow for rapid expansion of the repertoire of functionalities. Synchronization and data flow issues are addressed. The following measures have been taken wherever possible:

- Processes are data-triggered instead of time-triggered.
- Data is being pushed instead of pulled.
- Processes are started dynamically as they are requested.
- Processes enter an idle state (though not terminated) when they are not used.
- Resources are returned before entering the idle state.
- Memory consuming objects are only instantiated in the active time period.

A process-manager 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. The name is a unique identification key for the process and is used as reference by other processes in the system. This functionality is similar to the CORBA Nameserver [14]. If necessary, the process-manager will also start the requested system component.

The implemented system has been tested in a significant number of missions in a lab, where one room has been set up as an ordinary living room [1]. Work has also been initiated to make an implementation using CORBA [19].

Analysis This is a fairly modern system, where OO and base classes form a good basis for components. BERRA meets the hardware abstraction criteria. The interaction description is also addressed, although the particular choice of software (ACE), will probably not make the system widely used. No aspect of control or system description is considered.

Figure 1: A UML model of the implemented architecture

![UML Diagram]
4.3 OSCAR

In [5] a system architecture named OSCAR (Operating System for the Control of Autonomous Robots) is presented. It is based on CORBA and programmed in C++. Oscar has been implemented on a mobile robot that performs exploration of indoor environments. The concept of components and hardware abstraction are thoroughly addressed.

The basic module of OSCAR is the processing module (PM). A coordination module (CM) activates and controls the processing modules. Sensors and actuators are controlled by specific modules, in order to allow employment of different devices and robots. A processing module consists of a uniform frame and a processing core. The frame behaves like a server, accepting configuration and interaction requests, as well as data flow inputs. It implements several interfaces. The configuration interface serves as a receiver and executor of CM’s control commands. The generic interaction interface provides the possibility to visualize the current data together with other PMs for monitoring purposes. For data flow, two possibilities are implemented, pull and push. The processor interface provides data exchange for the processing core. Either via get (for unprocessed data) or supply (as output). The processing core has three different states: continuous mode, single step mode, and deactivated. State changes are only executed by the CM after a supply call. Continuous mode is applied together with push, and single step mode corresponds to push.

Analysis Referring to 2.4, the abstraction of hardware is taken into account. This is a modern system and one of the first to be based on CORBA. Therefore, the interaction description is well designed. Regarding control and system description, not much is mentioned in the work.

4.4 OPEN-R

OPEN-R [11] is an acronym for Open architecture for Robot entertainment. It is a proposed standard for interfaces of hardware and software components in entertainment robotics. Researchers at SONY have developed this for two reasons. The first is to promote research in intelligent robotics by providing off-the-shelf components and basic robot systems. The other is that they want to establish a new industry based on entertainment robots. The aim for the standard is to achieve system extension and reconfiguration capabilities for mechanical, electrical, and software components. The following features are suggested:

1. common interfaces for various components such as sensors and actuators.

2. mechanisms for obtaining information on functions of components and their configurations.

3. a three-layered architecture for hardware adaption, system services, and applications.

The software platform is based on Apertos, an object-oriented real-time distributed operating system. In [12], a reconfigurable robot platform is presented. By using configurable physical components, reconfiguration of the robot style (e.g. legs or wheels), as well as hot plug-ins, are enabled.

Analysis This is an interesting initiative and the only one here focusing on the hardware. Formal theoretical descriptions seem not to be the strong side of this work. The platform and OS will probably restrict it to the entertainment sector. Portability to other hardware is less likely, considering the company behind it has their own robot dogs to sell. The basic ideas should nevertheless be transferable to other areas within robotics.
4.5 Mobility

Mobility [15] is a robot integration software marketed by Real World Interface. Mobility provides Linux and NT support. The key features are (according to their marketing sheet):

- CORBA/IIOP communication
- CORBA 2.0 compliant ORBs for C++ and Java
- open architecture that enables development of new services or replacement of existing ones
- complete interface to all modules are fully specified
- all modules are fully multi-threaded and may be utilized as servers and clients

The basic services of Mobility include a Naming Service, Configuration Management, Hardware Servers, basic skills such as motion, mapping and navigation. Java client programs provide graphical user interfaces to the robot servers and interfaces. For a screen-shot of Mobility, see Fig. 3.

Analysis  This is probably the most modern style package in this survey. Certainly the interaction description is well covered. One serious drawback of the Mobility package is that it can only be obtained by buying one of the hardware robots sold by Real World Interface. Thus, any portability across platforms cannot be verified. As of the other issues mentioned in 2.4, we have no information to base any conclusions on.

4.6 TeamBots

TeamBots [3] is a Java-based collection of application programs and Java packages for multi-agent mobile robotics research. TeamBots is an example of a system with a high degree of granularity. A very large collection of classes and interfaces are available for developing new software. One selection of these classes is called Clay. Clay is a group of Java classes that can be combined to create behavior-based robot control systems. Clay takes advantage of Java syntax to facilitate combining, blending and abstraction of behavior. Clay can be used to create simple reactive systems or complex hierarchical configurations with learning and memory. A basic interface is inherited by all robot classes.
Figure 3: A screen-shot of the Mobility environment.
All the example programs run in a single process. However, multi-agent systems are the main interest of the author, so communication and synchronization between robots are well catered for. TeamBots is widely used in research and teaching. It has been ported to a number of robot platforms.

**Analysis** TeamBots is a popular package. It is very well designed and easy to learn. Simple applications can easily be constructed with the available class components. Hardware abstraction is provided. Interaction descriptions are achieved through java interfaces. Though not formalized, the base classes could form the basis for a control description. Through the use of Clay, the system description is addressed. The only question is scalability. The framework is excellent for small application, but will be too limited for a large complex system.

5 Conclusions

The amount of work being done for bringing robotics into the component philosophy, is definitely growing, although not yet very large. As has been seen, several key aspects are being addressed. Component specifications are studied at NIST, hardware interfacing is done at SONY. OSCAR and Mobility utilize modern software tools that simplify the component based approach. TeamBots is an example of a tight, cohesive (maybe not scalable) framework.

In several of the presented systems, the component concept is included in the design. However, the components are only considered to exist within the particular system and are exclusively tailored for that. Usage of such a component in another framework is then not possible without adaption. This adaption will in many cases be a too high price to pay for a potential user, and the component might well be abandoned for in-house development. It is also unlikely that the research community as a whole, would agree on a common architecture for conducting research.

What we need then is a common framework, with components that can be rearranged to form basically any architecture. Frameworks such as CORBA could well be an answer to this. The support of several programming languages and operating systems are important features.

Some researchers (as in many other areas), have thought that their field is so special, that standard solutions do not suffice. But now the field is maturing, and knowledge that has been available from computer science for quite some time is being adopted. Combining and expanding the efforts will pave the way towards the goal of having easily obtainable and reliable components for robot designers. To get there, the community must agree on a basic common framework. This framework has to encompass both analytical and practical aspects. A complete working reference implementation must be made to convince the researchers of the benefits of using it. The approach of the recent open source projects, combined with standards committees, may be a way of getting there.

References


