The Universal Robotic Service Architecture for Distributed Heterogeneous Robotic Systems

Han-Chieh Wei[1] Michael Nooner[2]

[1,2]Dept of Computer Science
University of Central Arkansas

Abstract

In this paper, we propose the Universal Robotic Service Architecture (URSA) with the goals of simplicity, adaptivity, and scalability. The architecture allows the system to interact with any type and/or different manufacture's robot systems. It also simplifies the application program by only need to focus on the applications and not have to worry about the communication and data mapping implementations.

Keywords: Distributed Robotic Systems, Distributed computing, Heterogeneous Robotic System.

1. Introduction

In robotic systems, there are varieties of tasks that can take advantages of using a group of robots to perform coordinated activities. These activities can not be completed by single robot either results from the spatial problem (multiple locations) or the temporal problem (moving speed). A distributed robotic system [5,6,8,9] considered to be heterogeneous if at least one member of the group is different from the others in one or more of the following attributes: mechanics, sensing or computing hardware, and computation methods [10]. For a heterogeneous robotic system, tasks can take advantages of different functionalities and facilities from the robot members which potentially provide more flexibility and fault tolerance compared to homogeneous groups. However, obviously the higher heterogeneity introduces higher complexity in interaction, data sharing, coordination, and scalability.

Most research of human-robot interaction in distributed robotic systems requires the already known system in terms of robot capabilities, communication protocols, and application program interfaces [1,2,3,4,9]. However, this requirement may not be achieved in large-scaled distributed heterogeneous robotic system in which there may be many different types of interaction and control mechanisms [11]. It is impossible for the application programs to implement all different types of interactions. Therefore, the interaction and control mechanisms have to be generic to provide general interaction with any robot systems.

In this research, we propose the Universal Robotic Service Architecture (URSA) which was designed with the goals of adaptivity, simplicity, and scalability in distributed heterogeneous robotic systems. The adaptivity allows the system to interact with any type and/or different manufacture's robot systems. The simplicity is for the developers focusing on the applications not worrying the low level communication implementations. URSA can be applied and scaled to different distributed robotic architectures.

The rest of the paper is organized as follows. Section 2 overviews the taxonomy of distributed robotic system architecture. Section 3 presents the Universal Robotic Service Architecture. The applications of URSA in distributed robotic architectures are discussed in section 4. Section 5 gives the conclusion the paper and future work of this research.

2. Distributed robotic system architectures

Several architectures have been proposed for distributed robotic system based on communication, control, computational and other capabilities. In this paper we categorize the robotic system according to the control and communication mechanisms.

Centralized Control

In the centralized control architecture, a central unit coordinates the group activities and makes group decisions.

Decentralized Autonomous Control

In *autonomous* control architecture**,** each robot has the full control of its activities**.** An example of true autonomy can be found in emergent behavior studies. These studies use low-power robots that perform simple commands in response to environmental conditions. Most of these robot have little computational power, so many of the environmental responses must be calculated somewhere else.

Hub Control

Under hub control there is a group of robots that in a similar manner as they did in decentralized control, we call these robots the queens. And there is a second group of robots, categorized as low-power robots, act as drones and are commanded by the queen. For example, say we wished to have our robots map an area that would put them out of signal range of a central data consumer. These robots then could trail behind them a set of signal boosting robots. The queen then would leave them behind as it left the signal range each dropped drone**.**

Hybrid

In practice, many systems do not conform to a strict centralized/decentralized architecture. Also, notice that none of the above approaches are mutually exclusive**.** Therefore, one of the servers in the decentralized approach could be a central authority. Likewise the queens from the hub approach could forward the drone's data to other queens or a central authority, and so forth.

3. The Universal Robotic Server Architecture

There are many manufacturers of robots. Each uses its own proprietary drivers for communication between client programs and the physical robot. This is understandable from the manufacturers standpoint, but creates complexity for application development. It would be desirable to interact with all the robots in the same manner. This idea is the driving force behind the development of the Universal Robotic Service Architecture (URSA). URSA is designed with a similar philosophy as XML. This means that a programmer can interact with a robot without needing to know in advance the robot's type and manufacturer.

This system basically boils down to a client-server architecture. In order to maintain the simplicity of programming for the robot, the end programmer can develop applications interacting with proxy robot objects and should never have to write code for remote connections. On the robot side, a driver (or adaptor) is required for each robot such that the server can interact with robots in a standard manner.

Figure 1 illustrates the overall architecture of URSA. For the purposes of our discussion, a server is noted as the URSA server. The term data consumer or just consumer denotes a program written that receives data from a robot and processes it. A commander is some process or user that gives commands to the robot (note that a commander may be a separate entity from the data consumer). A robot manager is a process that controls a particular robot's driver and handles communication between the server and the robot.

URSA is designed with the black box mentality. There is a collection of processes which is called the URSA server, they act as a hub for robot-to-consumer and commander-to-robot communication. This server is opaque, the robots and the consumers need to only know its address. The design of URSA is based on the following assumptions. First, the robots can be controlled wirelessly. The server, the consumer, and the robot manager can all reside on the same machine that is directly connected to the robot. This assumption helps us to use robots that use WiFi as their means of

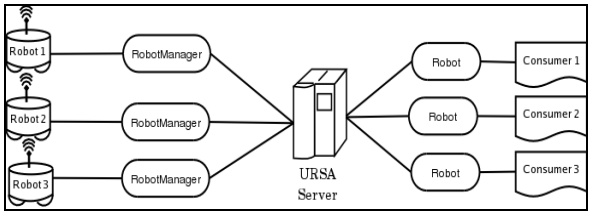

Fig.1: Overall architecture

communication. Secondly, we assume that a robot can be defined as a process that generates data and takes zero or more commands. By this definition, the robots can be (1) mobile robots equipped with sensors and embedded with capability of autonomous mobility; (2) immobile sensors embedded with sensing and communication. The third assumption is that there are two types of robots. Namely, there are low-power robots that have limited computational capacity and are always controlled remotely; and high-power robots that have similar computational capacity as a mid-range laptop. This assumptions is based on the fact that a heterogeneous robotic system consists of low-power and/or high-power robots.

What these assumptions lead to is a highly distributed design. The robot managers communicate with the server through the network. Likewise the consumers and commanders, via the Robot object, communicate with the physical robot through server, which is remote. All commands and data pass through the server. There is no direct communication between the robot and the consumer/commanders.

4. Architecture Components

In this section, we introduce the architecture components from three perspectives: applications, robots, and URSA server.

4.1. Application Programs

From the programmer's perspective, URSA is designed with a similar philosophy as XML. XML really is just a description of data. It is useless unless your program knows that particular data's structure. The same applies for URSA. While URSA allows an

application to interact with any robot, the application needs to know the capabilities of the robot it interacts with. This information can be collected when the user requests a connection to a given robot from the URSA server. Once the connection is built, an XML document (listed in [7]) is returned listing all the data that a robot can generate and all the commands a robot can take.

The purpose of this document is three fold. First, it allows a custom *Robot* object to be built at runtime on the users machine, more about this described in the next section. Secondly, it allows the programmer to create dynamic programs that can detect and take into account the different capabilities of different robots. Lastly, the programmer or commander can inspect this document to learn the capabilities of the robot before development or issuing commands.

For simplicity, application programs interact with robots through local Robot object which is the proxy of the remote robot. To create a Robot object, the program asks the server for a specific robot. The server then returns the XML document associated with that particular robot. The Robot object is created based on the specifications in the XML document. The program then interacts with this object as if it were the actual physical robot.

4.2. Robots

The robot side, as shown in figure 3, consists of three components: *robots*, *DataDriver*, and *RobotManager*. The DataDriver is really a set of API's which serves as an adaptor for the robot. The *RobotManager* object handles robot-to-server and server-to-robot communication, i.e. the object hands the driver the commands, and transmits the data from the driver to the server. Basically it is the job of the RobotManager to interact with the driver. For each type of robots, an XML document is created to specify the capabilities of the robot. This XML document will be transmitted to the server in the initial handshake, and latter on to the consumers.

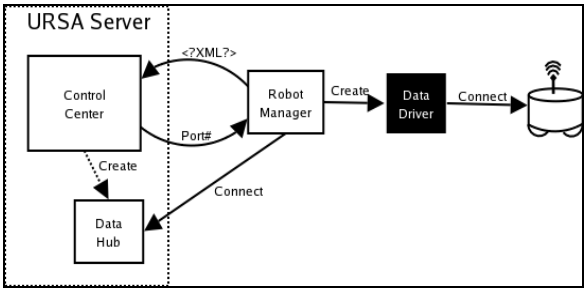

Fig. 3: Robot Connection

4.3. URSA Server

The URSA Server is an opaque entity that both the robot managers and the robot objects interact with. The URSA Server allows data to flow from the physical robot to the data consumers in a timely manner. More importantly, commands are routed to the robot with minimal delay. In the following, we give a description how the server connects robots and consumers.

In the URSA server, as shown in figure 3, the *ControlCenter* process will be started first. This process acts like a central server that all connections initially connect to. Next, a *RobotManager* process must be started for each robot to be served. The RobotManager makes sure the DataDriver gets the data from a robot. It handles the initial connection to the ControlCenter, forwards data to a DataHub and, finally, forwards commands to the DataDriver. The

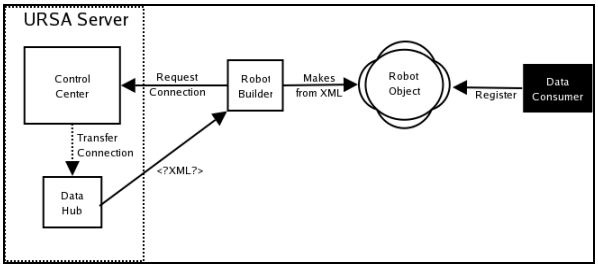

Fig. 4: Consumer Connection

RobotManager communicates to the server remotely. When the RobotManager starts up, it first waits until it can get a connection to its robot through DataDriver. Then, it connects to the ControlCenter and sends the XML document of the robot. The ControlCenter responds by creating a *DataHub* object for each robot. The DataHub acts similarly to an Ethernet router. All commands and data are routed through this object.

To connect to the consumer, illustrated in figure 4, the ControlCenter accepts connection request from consumer through Robot object and hands the connection to the DataHub of the robot the consumer wishes to connect to.

An example for complete data and command flow can be found in our previous research [7].

5. Applications of URSA

There are several advantages of this architecture. First, the users or applications can interact with any given robot as if they all use the same drivers. By implementing drivers that are independent from the manufacturer, we can fake the capabilities of a robot in order to meet some criteria. The second advantage is that the work load of the programmer is drastically reduced. The implementation for communication and driver are taken over by the system. Thirdly, the URSA system can be easily scaled. Robots can join

the system at any moment once it has the DataDriver and XML document ready.

Furthermore, the URSA system allows for the creation of a heterogeneous robotics simulator (currently under construction). It allows users to program and simulate runs without having to use the actual robots. This is important especially if users want to write a program that allows for cooperative efforts among heterogeneous robots.

As presented in section 2, URSA can be easily adopted to these architectures:

Centralized Control URSA lends itself well to this architecture. A data consumer can register itself with multiple Robot objects. This allows the data consumer to act as an authority over multiple robots. Also commands are prioritized, so that an authority, e.g. a human user or a process, can interrupt the command flow if it has a higher priority. Further this can create a hierarchy of commanders.

Autonomous Control Under URSA it is simple to implement this approach by having different consumers/commanders for each robot.

Hub Control Under URSA the queen robot acts as the server and consumer for the drones. Furthermore, the queen acts as a server, consumer, and robot for the other queens, i.e. similarly to decentralized control [5].

Hybrid For this type of architecture, one of the servers in the decentralized approach could be a central authority. Likewise the queens from the hub approach could forward the drone's data to other queens or a central authority. And so forth. This shows that URSA is ultimately a flexible architecture for manipulating multiple heterogeneous robots.

6. Conclusion and future work

In this paper, we propose the Universal Robotic Service Architecture (URSA) for distributed heterogeneous robotic systems. This design solves the problem when multi-user access multi-robot simultaneously [11]. It also simplifies the application development in the heterogeneous robotic systems where all the robots can be treated the same. Furthermore, URSA can be adapted to different types distributed robotic system architectures.

For the future work, we plan to apply URSA to create a robotics simulator. This simulator will simulate any given robot described by the XML document mentioned above.

7. References

[1] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. “An architecture for autonomy.” *Int. J. of Robotics Research*, 17(4):315-337, April 1998.

[2] R. C. Arkin, T. Balch, “Cooperative Multiagent Robotic Systems,” in: D. Kortenkamp, R. P. Bonasso, R. Murphy (Eds.), Artificial Intelligence and Mobile Robots, MIT/AAAI Press, 1998.

[3] E. Coste-Maniere and R. Simmons. “Architecture, the Backbone of Robotic Systems,” *In Proc. of the IEEE Conference on Robotics and Automation*, San Franciso CA, April 2000.

[4] R. Fierro, A. Das, J. Spletzer, R. Alur, J. Esposito, Y. Hur, G. Grudic, V. Kumar, I. Lee, J. P. Ostrowski, G. Pappas, J. Southall and C. J. Taylor, "A framework and architecture for multi-robot coordination," *The International Journal of Robotics Research*, vol. 21, no.10-11, pp. 977-995, Oct-Nov 2002.

[5] A. Howard, L.E. Parker, and G.S. Sukhatme. “The SDR experience: Experiments with a Large-scale Heterogenous Mobile Robot Team.” *In Proc. of the International Symposium on Experimental Robotics*, 2004.

[6] Bradley Kratochvil, et al.: “Heterogeneous Implementation of an Adaptive Robotic Sensing Team.” *Proc. of the 2003 IEEE International Conference on robotics & Automation*. Taipei, Taiwan, 2003.

[7] M. Nooner and H.C. Wei. “The Universal Robotic Service Architecture,” in Proc. of 3rd Annual Consortium for Computing Sciences in Colleges Mid-South Conference, 2005.

[8] L. Parker. “Alliance: An architecture for fault tolerant cooperative control of heterogeneous mobile robots.” In *Proc. of International Conference on Intelligent Robots and Systems*, 1994.

[9] Lynne E. Parker. “Current State of the Art in Distributed Autonomous Mobile Robotics.” In L. E. Parker, G. Bekey, and J. Barhen, editors, *Distributed Autonomous Robotic Systems*, vol.4, pp.3–12, Springer, Tokyo, 2000.

[10] P. Rybski, et al. ”System Architecture for Versatile Autonomous and Teleoperated Control of Multiple Miniature Robots.” *Proc. Of the 2001 IEEE International Conference on Robotics and Automation*, Seoul, Korea, 2001.

[11] Ashley D. Tews, Maja J. Matarić, and Gaurav S. Sukhatme. “A scalable Approach to Human-Robot Interaction.” *Proc. of the 2003 IEEE International Conference on robotics & Automation*. Taipei, Taiwan, 2003.