Chapter 1
Transparent multi-robot communication exchange for executing robot behaviors

Carlos Agüero and Manuela Veloso

Abstract Service robots are quickly integrating into our society to help people, but how could robots help other robots? The main contribution of this work is a software module that allows a robot to transparently include behaviors that are performed by other robots into its own set of behaviors. The proposed solution addresses issues related to communication and opacity of behavior distribution among team members. This location transparency allows the execution of a behavior without knowing where is located. To apply our approach, a multi-robot distributed receptionist application was developed using robots that were not originally designed to cooperate among themselves.

1.1 Introduction

Service robots help humans in many tasks, including cleaning tasks, medical applications, surveillance and guiding. Service robots solve a specific problem and consequently their sensors and actuators are designed accordingly. A natural consequence of this design is that these robots are not reusible in other tasks they were not originally conceived for.

We find that these robots are very good at performing the task for which they have been developed; we could say that they are experts in one or more specific tasks. This specialization can be understood as an ability to perform an action.

As new applications appear, humans demand more functionality from service robots. One way to meet these challenges is to expand the capabilities of the robot, whether in hardware or software. Instead, we explore another approach for the robot to cooperate with other robots that are able to solve the new task demands. The latter

Carlos Agüero
Universidad Rey Juan Carlos, Camino Molino, 28943, Madrid,Spain e-mail: caguero@gsyc.urjc.es

Manuela Veloso
Carnegie Mellon University, 5000 Forbes, Pittsburgh, PA 15213, USA e-mail: veloso@cs.cmu.edu

approach requires a shared effort among all the robots, since we have gone from working with a single robot to a robot team. Quoting Doug McIlroy, the inventor of Unix pipes, "This is the Unix philosophy: Write Programs that do one thing and do it well. Write Programs to work together"[11].

In this paper we present an approach that helps to make available to the whole team all the different individual behaviors of each robot for solving tasks. This aggregation is done transparently, so that all the robot share the same set of behaviors. These behaviors could be performed by the robot itself or they could require cooperation with other robots that would run them in their place. The net effect is that the behavior is done opaquely, regardless of which of the robots have completed it.

The modular solution described in this work addresses issues of communication between robots with different operating systems, programming languages and integration with the host infrastructure on which it needs to operate.

The rest of this paper is organized as follows. First, we discuss the related state of the art. Then, the design of the software module and its main components are presented. Next, we detail the distributed receptionist application developed using our approach. Finally, some discussion and future lines are presented.

1.2 Related work

As we witness the explosion of cloud computing, the idea of using other resources on the Internet has spread to robotics with the creation of Cloud Robotics[7]. This approach provides virtually unlimited resources alleviating the limited features of robots. For example, the Google Goggles[6] application allows the user to send a photograph of an object and, if it has been previously processed by someone else, the object is recognized. The cloud can also store knowledge and models. The RoboEarth project[12] describes how an articulated arm equipped with sensing capabilities can create a model to open a drawer. Then, another articulated arm with rudimentary sensors can request the information previously stored in the cloud and use it to open the drawer by adjusting the model to its actuator skills.

The key idea of reusing knowledge is fundamental to our approach and a prevailing concept in Cloud Robotics. However, the module we present here also aims to reuse the behaviors of other robots, reaching a higher level of cooperation based solely on knowledge reuse from the cloud.

The term *Robot as a Service*[2] was created using the concept of *Service-Oriented Architecture*, which provides a communication mechanism through standard interfaces and standard protocols. The idea that each robot maintains a common layer for offering services is shared with the work presented here. However, our approach seeks to make transparent the fact that the services may require communication with another robot to get started, achieving an even higher level of abstraction.

Swarmnoids project[3] is an example of how a team of heterogeneous robots can solve tasks that are beyond their individual capabilities. UAVs (Unmanned Aerial Vehicles) cooperate with ground vehicles in highly-coupled tasks, generating self-

organized cooperative behaviors. The work presented in [9] for the treasure hunt domain uses the very popular market-based mechanism of coordination to determine which robot performs each behavior in an environment that requires the cooperation of heterogeneous robots. In [4] other work is described based on this principle. For a more detailed analysis and a Multi-Robot Task Allocation taxonomy consult [5]. In all of these works, communication between team members is made explicitly, losing the level of transparency that our novel alternative offers.

1.3 Multi-Robot Task Module

Multi-Robot Task Module (MRTM) follows a distributed approach and, accordingly, each robot should run an instance of MRTM. The main objective of the MRTM is to encapsulate all access to the behaviors the team can offer. The behaviors do not necessarily have to be implemented within the module itself, but the interface to access them does. Some interesting features offered by this design are the behavior location transparency and its cross-platform availability.

The behavior location transparency concept is defined as the ability to hide which robot is providing a behavior for solving a given task. In practice, this solution allows us to maintain a robot team, where all robots share a single interface. In addition, this interface brings together all the behaviors that each robot can perform. Essentially, the work of the MRTM module is to invoke the behavior that has been requested, either by running an algorithm or a behavior on the robot itself, or by requesting the behavior of another robot's MRTM component. Figure 1.1 shows the internal structure of the MRTM, whose main components will be explained below.

When there is a robot team where all members have a different behavior repertoire, it is necessary to know which robot can perform which behavior. A possible solution is that each application queries all the robots, and obtains a list of available behaviors, along with the robot that provides that behavior. Location transparency takes care of this tedious operation and delegates all the job to MRTM module, so applications just use the behaviors without knowing where they actually reside.

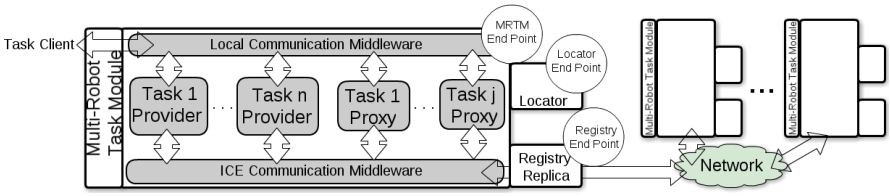

Fig. 1.1 Overview of the Multi-Robot Task Module

Ice Communication Middleware

All communication among robots is performed by Internet Communications Engine[8] (Ice). Ice is an object-oriented middleware used in distributed systems. Ice's main virtues include its multi-platform and multi-language support, and efficiency.

An Ice object is an entity in a local or remote robot that can reply to client requests. An interface declares a set of operations (behaviors) that an Ice object can perform and clients can invoke. Operations performed between client and server are declared using an independent of a specific programming language called Slice.

A proxy is an object used to contact with a remote Ice object. The proxy emulates a specific Ice object's interface and its operations can be invoked. In MRTM we use indirect proxies, which are composed by an Ice object identifier and object adapter identifier. An object adapter is a container of Ice objects in a given robot. An indirect proxy does not contain any addressing information. To find the correct server, the client-side Ice run time passes the proxy information to a location service. In our system, to eliminate this single point of failure and ensure the highest possible availability, we have included a *Registry* for each MRTM, thanks to the master-slave replication supported on Ice.

Ice Communication Middleware (ICM) combines all operations related to Ice. These operations are the creation of the object adapter and the registration of Task Providers within the adapter. In addition, this sub-module is responsible for resolving indirect proxies to perform remote invocations and receiving calls from other robots, as well as initializing the registry process.

In MRTM, an Ice object is realized by creating a class that implements the interface to that Ice object. In our approach these classes are called Task Providers and consist of related behaviors for a given task.

To receive requests on a given MRTM, they must be called by a different MRTM. When a particular behavior is requested, if it cannot be executed by any local Task Provider, the corresponding Task Proxy is used. The proxy is just a binding in the form of an object, which hides the communication with another MRTM. To invoke the method of the object that triggers the behavior, the proxy has to be resolved. This is to find the End Point for the object adapter that includes the Task Provider of interest. *Locator* is responsible for implementing this work. *Locator* communicates with one of the *Registries* and resolves the indirect proxy. Once the proxy is set up, the remote invocation can be performed and will be received by the ICM's object adapter of another robot. In turn, this object adapter will distribute the request to one of its Task Providers, which will launch the behavior.

Local Communication Middleware

When a client or application invokes a specific behavior, MRTM receives the request through the Local Communication Middleware (LCM) sub-component. The implementation of this sub-component depends on the internal communication system of the robot. For example, using ROS[13], this sub-component would be a node

on the robot. The interface of this LCM node would form a set of ROS services. In short, the function of this sub-module is to integrate into the host and serve as an infrastructure gateway for clients of behaviors. There are two steps to implement this component: To instantiate an MRTM object and to create an interface to use all the behaviors that the robot team is able to perform. Depending on the local middleware this could be a set of methods declared in a header file, a series of services, or a series of published topics.

Task Providers

The Task Provider is another sub-component of the MRTM. Its mission is to trigger the execution of a particular behavior on the robot. Inside MRTM, there will be as many Task Providers for the local behaviors that the robot can perform. MRTM does not intend to modify the internal architecture of the robot and make it monolithic. It is the opposite, as these local behaviors implementations would remain independent modules distributed within the robot (e.g., other ROS nodes). MRTM simply groups all Task Providers to manage calls to these local behaviors more easily.

Task Proxies

Like the Task Provider sub-component, which start the execution of local behaviors, the Task Proxy sub-component run behaviors remotely requiring execution by another robot. To implement each Task Proxy you must get an indirect proxy to MRTM that is capable of performing the behavior, and then, to perform the remote invocation using the indirect proxy as an object and the behavior as a method.

1.4 Multi-Robot receptionist

To illustrate the MRTM design made in section 1.3, we applied the proposed MRTM to a team of robots performing a distributed people reception task. The task required a receptionist to welcome visitors and ask for the name of the person to visit. The receptionist retrieves and offers the office location associated with that person and suggests the option of being escorted to the office.

To conduct the experiment, two different robots and a directory program that ran on a conventional computer were used. The first is the Nao robot[1]. It is a medium-sized humanoid robot running GNU/Linux. Nao features two cameras, Wi-Fi, bumpers and hi-fi speakers, among other sensors and actuators. Its mission in the experiment was to meet the people who came and to offer directory and escort tasks.

The second member of the team was the CoBot robot[10]. It is equiped with a LIDAR, two cameras, one kinect and an omnidirectional base. CoBot is a visitor companion robot able to navigate through the environment, transport objects, deliver

messages, and escort people. Its goal in this experiment was to escort a visitor from a known area (the exit of the elevator) to the office of the person he wanted to visit.

The third and final team member was a directory program that ran on a laptop. It contained a directory of people and their associated offices.

Figure 1.2 shows the overall architecture of the experiment (the elements *Registry* and *Locator* have been deliberately omitted to simplify the diagram). All agents have the same interface, which allows the deployment of directory and escort behaviors. While in this experiment the application of user interaction was only deployed on the Nao, any other robot could technically have offered the same functionality.

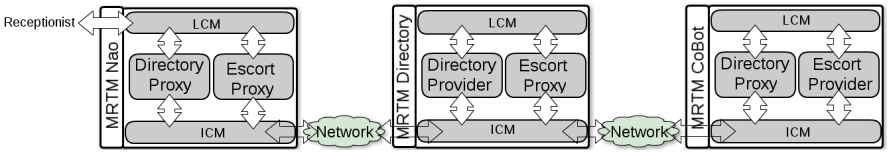

Fig. 1.2 Overview of the MRTMs on the Receptionist experiment

The LCM sub-component of the CoBot robot was integrated into a specific ROS node responsible for managing the behavior of the robot. In turn, the LCM modules of the Nao receptionist robot and directory program were included in objects that communicated with each host infrastructure through regular method invocations.

The Slice interfaces shared among the agents of this experiment are shown below. The *Directory* consists of a single function that returns the office associated with a person. In turn, the *Escort* interface includes the method *escort*, which selects a specific office and a time window and returns a task ID. This value can be used to cancel the booking in a future calling to the *cancelTask* method.

```
module MultiRobotReceptionist {
 interface Directory  {
   string people2office( string personName ); };
 interface Escort {
   string escort( string room, string startDate, string startH, string startM,
       string startP, string endDate, string endH, string endM, string endP );
   int cancelTask( string taskId ); }; };
```

After greeting and interacting with the visitor, Nao uses its Directory Proxy to consult the directory program and ask for the destination. Note that the Nao does not even know which robot or agent actually implements this service. The receptionist application invokes the method *person2office* as if it is implemented in itself. Once the visitor is informed of the office number, Nao offers the visitor the option of being escorted to the office. If the proposal is accepted, Nao uses the *escort* method offered by its MRTM to start escorting the visitor.

As we described before, the MRTM module takes over and triggers the escort behavior using the Escort Proxy of the CoBot robot. CoBot has a scheduler for requesting different behaviors and when they should be executed by the robot. The Escort Provider of the CoBot robot makes a reservation for the escort task before offering visitors the option of being escorted. If the reservation is successful, the receptionist

Nao offers the possibility of escorting the visitor. Then, if the visitor rejects the offer, Nao uses the *cancelTask* method to remove that task from the scheduler. Figure 1.3 shows a sequence of frames illustrating the key moments of the experiment.

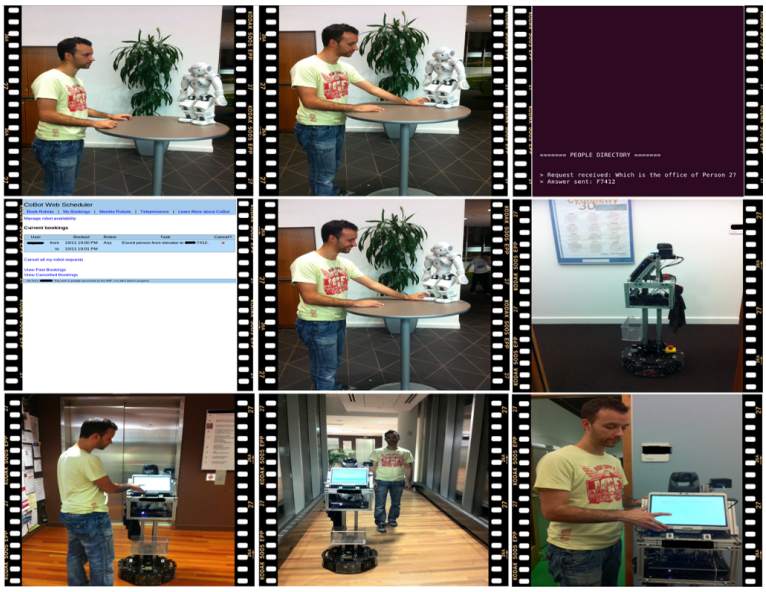

Fig. 1.3 Sequence of shots extracted during an experiment escorting a person. Upper left: Visitor is greeted by the Nao receptionist. Upper central: Visitor and Nao interact using voice and bumpers for select a person. Upper right: Directory program receives the request and answers with the office number. Middle left: The escort task is reserved on the CoBot scheduler. Middle central: Visitor accepts to be escorted. Middle right: CoBot starts navigating towards the meeting spot. Lower left: CoBot waits at a known location until the visitor arrives. Lower central: The visitor is escorted by CoBot. Lower right: The visitor arrives at the destination's office.

The resources consumed by MRTM were measured during the experiment. CPU overhead was completely negligible, 40MB of memory was consumed and there was no continuous bandwidth used. The spikes on the bandwidth consumption were at a maximum of 1Kb/sec. and occurred during the Directory and CoBot requests. A video of the experiment can be downloaded at this link[1].

1.5 Conclusions and future work

We have presented the MRTM as a solution to the problem of the remote execution of behaviors with location transparency. All the work for localizing behaviors and interoperability with different platforms has been solved using the tools provided by

[1] http://dl.dropbox.com/u/2831576/PAAMS12_caguero.mov

the Ice middleware. This is one of the advantages over other middleware for robots. Our approach allows for the integration of a MRTM module into more devices such as robots with different operating systems or even mobile phones, with a very low impact on CPU overhead, memory consumption and network bandwidth.

An experiment with robots and humans was conducted successfully for a multi-robot receptionist task. The receptionist program running on the Nao robot was ignoring where was the people directory located or which robot was escorting the visitors. All the low level details of exchanging information were hidden by MRTM.

Thanks to the common interface provided by MRTM, programs that use behaviors are easier to design because there is no explicit negotiation among the robots.

Despite the results obtained, there are open questions for future research. One of the main future lines is the inclusion of a Multi-Robot Task Allocation sub-module used by multiple robots that provide the same Task Provider. It also could be responsible for negotiating with multiple robots to choose the most appropriate in terms of some utility function.

Acknowledgments

The authors also would like to thank all the members of the CORAL group, Robotics Institute and URJC Robotics group who are collaborated in this work.

References

1. Aldebaran Robotics. http://www.aldebaran-robotics.com, 2011.
2. Y. Chen, Z. Du, and M. García-Acosta. Robot as a Service in Cloud Computing. In *Proceedings of the 2010 Fifth IEEE International Symposium on Service Oriented System Engineering*, SOSE '10, pages 151–158, Washington, DC, USA, 2010. IEEE Computer Society.
3. L. G. F. Ducatelle, G. Di Caro. Cooperative Self-Organization in a Heterogeneous Swarm Robotic System. In *Proceedings of the Genetic and Evolutionary Computation Conf.*, 2010.
4. B. P. Gerkey and M. J. Mataric. Sold!: auction methods for multirobot coordination. *IEEE Transactions on Robotics*, 18(5):758–768, 2002.
5. B. P. Gerkey and M. J. Mataric. A formal analysis and taxonomy of task allocation in Multi-Robot systems. *The International Journal of Robotics Research*, 23(9):939–954, Sept. 2004.
6. Google. Google Goggles, http://www.google.com/mobile/goggles/, 2011.
7. E. Guizzo. Cloud Robotics: Connected to the Cloud, Robots get Smarter, http://spectrum.ieee.org/automaton/robotics/robotics-software/cloud-robotics, 2011.
8. M. Henning. A New Approach to Object-Oriented Middleware. *IEEE Internet Computing*, 8:66–75, January 2004.
9. E. Jones, et al. Dynamically Formed Heterogeneous Robot Teams Performing Tightly-Coordinated Tasks. In *Int. Conf. on Robotics and Automation*, pages 570 – 575, May 2006.
10. S. Rosenthal, J. Biswas, and M. Veloso. An effective personal mobile robot agent through symbiotic human-robot interaction. In *AAMAS 2010*, volume 1, pages 915–922, May 2010.
11. P. H. Salus. *A quarter century of UNIX*. ACM Press, New York, NY, USA, 1994.
12. M. Waibel et al. RoboEarth. *Robotics Automation Magazine, IEEE*, 18(2):69–82, 2011.
13. Willow Garage. ROS (Robot Operating System), http://www.ros.org/, 2011.