ROCI 2: A Programming Platform for Distributed Robots based on Microsoft's .NET Framework

Vito Sabella, Camillo J. Taylor, Scott Currie
GRASP Laboratory
University of Pennsylvania
Philadelphia PA, 19104

Abstract

This paper describes the ROCI 2 system, a framework for programming distributed teams of robots. A programming model for such ensembles is proposed and the implementation of this model within the .NET framework is described.

Introduction:

As sensors, actuators, microprocessors and wireless networks become cheaper and more ubiquitous it has become increasingly attractive to consider employing teams of small robots to tackle various sensing and manipulation tasks. Consider, for example, the team of "ClodBuster" robots shown in Figure 1 engaged in the process of locating and manipulating a box cooperatively. Working in tandem, the robots are able to perform tasks that they could not achieve independently.

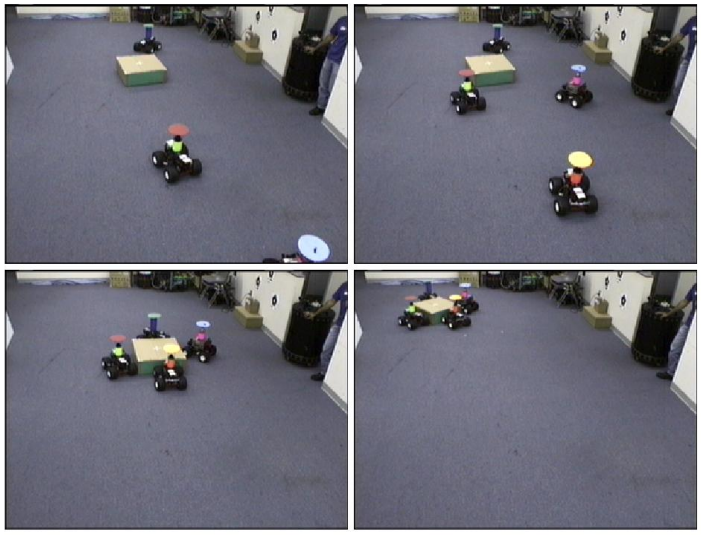

Figure 1. Team of ClodBuster mobile robots engaged in cooperative localization and manipulation task

In order to exploit the capabilities of robot teams like this one we need to develop effective models and methods for programming distributed ensembles of sensors and actuators. This paper describes a platform for programming these types of systems which leverages the development tools, libraries and infrastructure being developed for Microsoft's .NET framework.

In this approach the .NET framework is used to implement functionality that is rapidly becoming commonplace in the GRID computing and agent communities, capabilities like web service description and remote method invocation. The .NET framework builds upon established standards such as XML (Extended Markup Language) and WSDL (Web Service Description Language) which can be used to describe and publicize robot and agent capabilities. It also supports SOAP (Simple Object Access Protocol) and has built in capabilities for routing and handling service requests over a network using HTTP. Also important is the fact that the .NET framework defines a device independent intermediate language, MSIL, analogous to Java bytecodes which gives us a convenient mechanism for shipping executable content around a network.

Programming Model:

The distributed networks of sensors, processors and actuators described in the previous section require a radically different programming model than the one employed for most traditional robotic applications. In most situations the programmer is faced with the task of developing software for a single processor interacting with a prescribed set of sensors and actuators. He or she can typically assume that the configuration of the target system is completely specified before the first line of code is written.

In contrast, when developing code for our robot teams, we must account for the fact that the number and type of robots available at runtime cannot be predicted. We expect to operate in an environment where robots will be added and removed continuously and unpredictably. Further, we must expect an environment where the robots will have heterogeneous capabilities; for example, some may be equipped with camera systems, others with range sensors or specialized actuators, some agents may be stationary while others may offer specialized computational resources. This implies that the program must be able to identify and marshal all of the resources required to carry out the specified task automatically.

One of the core ideas in ROCI 2 is that programming an ensemble of robots should resemble writing a *virus*. A particular task may involve several robots with heterogeneous capabilities interacting in a particular way. Much as the DNA of an organism contains a blueprint that allows us to build an individual out of cells which differentiate to perform different roles – our robot program would actually consist of several programs which would specify how different robots in the team should behave and interact. When this program is injected into a network it would automatically start to recruit robots to perform various roles in the program.

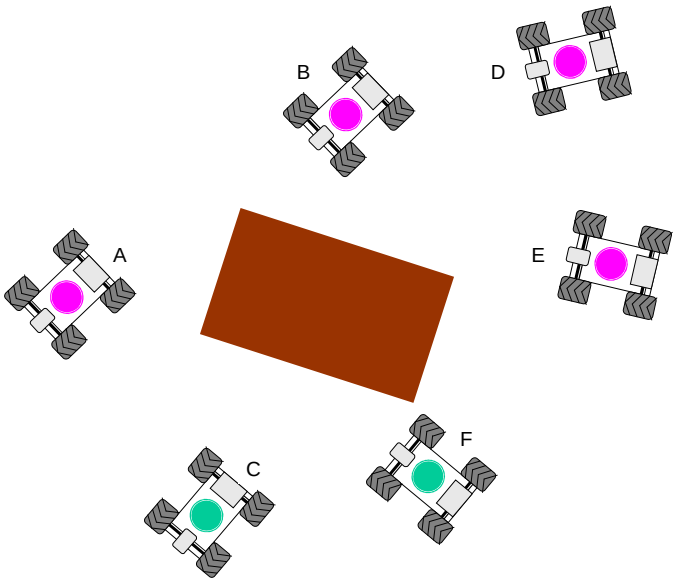

Figure 2: A program inserted into the network is responsible for marshalling all of the required resources. In this example, the program initially infects robot A which recruits robots B and C who in turn recruit robots D, E and F.

In Figure 2 robot A was originally “infected” with the robot program that the user wants to execute. The program carries a complete description of the number of robots required to carry out the task and the required capabilities. Based on this description, robot A recruits robots B and C who in turn recruit robots D, E and F. If one of the robots should fail, the other agents interacting with it would notice the failure and recruit another robot with appropriate characteristics to fill its place in the team. Just as multi-cellular organisms develop from embryos based on the instructions in their DNA, our robot teams should self-organize based on the programs inserted into the network.

In order to implement this programming model, the underlying system should provide the following capabilities: the individual robots must be able to advertise their capabilities to each other in such a way that a robot program can discover which systems are available for use. The system must also provide a mechanism for distributing executable content so that a program can recruit members to perform roles. The next section describes how these capabilities are implemented in ROCI 2.

Implementation:

Our current implementation of ROCI 2 is written in C# and based on the .NET framework. The core component in our implementation is the ROCI 2 kernel which mediates many of the interactions between the robots in the team. Every node in a ROCI 2 network runs a copy of this kernel which is responsible for handling networking, Inter Process Communication and access control.

In addition to the kernel, each robot may be running several processes which encapsulate various services and capabilities offered by this node. For example, the ROCI system developed for our robotic blimp ran processes which encapsulated the functionality of the onboard video camera, the onboard GPS sensor and the motion control system. Each of these processes can expose its interface to other processes in the network through the ROCI 2 kernel.

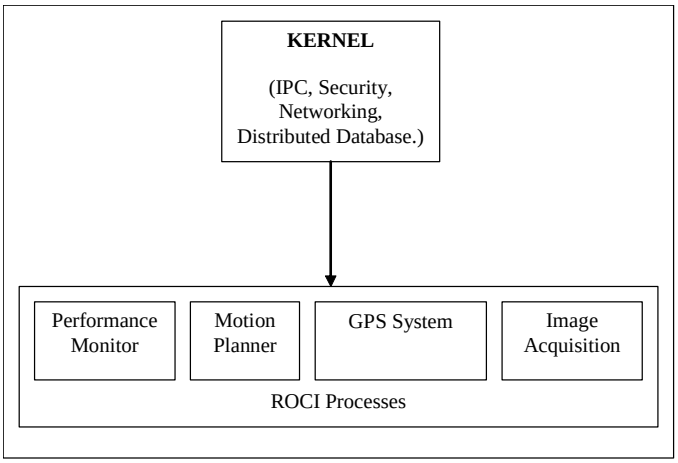

Figure 3: In the ROCI framework the ROCI kernel mediates, process creation, network access, inter process communication and database issues. Other processes encapsulate the services and capabilities provided by this robot.

Each robot is responsible for maintaining its own database of nodes in the network and their associated services and capabilities. This database is continually updated as the robot interacts with other nodes in the system. This implementation offers several important advantages. Firstly, it allows us to fully distribute information about the network over all of the nodes in the system so there is no central repository to maintain and defend. Secondly, it means that robots and services can be added and deleted from the network dynamically since the robots eventually correct and update their databases as they interact. Entries in this database are encoded in XML and the robots can query the system to discover nodes with particular attributes in the usual way.

Once a process associated with a resource on a particular node has been identified in the database, a robot can interact with that resource through the inter process communication mechanisms provided by the ROCI kernel. The ROCI 2 kernel allows any pair of processes in the system to communicate via events or streams. Events are extended by the system so that they can seamlessly traverse the network and streams are dynamically created by the kernel in response to connection requests.

The ROCI kernel also accepts requests for creating new processes from the network. Once a request is received the kernel transfers the executable content from the client and then performs a security check to ensure that the client has the appropriate permissions.

In the current implementation, executable content is distributed in the form of serialized Microsoft Intermediate Language (MSIL) objects. MSIL provides us with a device independent approach to specifying the desired functionality. It also allows us to consider implementing finer grained security policies using the run time execution environment.

The ROCI system also provides mechanisms for device abstraction so that programs written to access devices such as video cameras and GPS systems can be executed correctly on a variety of hardware platforms.

Developing within the .NET framework affords us a number of benefits. Firstly, the framework provides mechanisms for accessing and publishing resources and services provided over a network. The demonstration application developed for the robotic blimp shown in Figure 3 made use of the web services provided by MapQuest to download and display maps corresponding to the readings returned from the onboard GPS sensor. Similarly, we envision having ROCI processes expose their interfaces to the environment as web services so that remote users could directly interact with sensor and actuator processes through conventional browsers.

Figure 3: Applications developed using the ROCI 2 framework have been deployed on the robotic blimp platform shown here.

The framework also makes it simple to access the functionality provided by DLLs. This capability allows us to make use of standard libraries and device drivers in a natural way. For example, some of the applications that have been developed with this framework make use of built in media encoders to transmit video and audio information between ROCI 2 nodes. This mechanism also allows us to make use of specialized code such as image processing libraries and Matlab routines which can be packaged as DLLs.

The .NET framework also allows us to target embedded, real time operating systems like Windows CE and to develop code for smaller devices like the Compaq IPAQ. This capability is particularly valuable in the context of teams of mobile robots since it allows us to develop clients that can be used to access and task the robot ensemble from handheld devices.

Conclusions:

The ultimate goal of the ROCI 2 project is to develop a programming framework that will allow us to use existing compilers, debuggers and web browsers to rapidly prototype applications for our robotic ensembles. We expect that we can spare ourselves a lot of development effort by riding on top of existing commercial frameworks and extending them in ways that are useful to us.

By adopting a viral programming model we hope to be able to develop programs that automatically configure the available nodes in the network to perform the user's task. This programming model is relatively easy to implement within the .NET framework which provides many of the required capabilities for developing network aware programs.

Current work focuses on a number of issues related to authentication, process priority and resource allocation which are particularly interesting in the context of distributed robotics.

References:

- D. L. Martin, A. J. Cheyer, and D. B. Moran, "*The Open Agent Architecture: A framework for building distributed software systems*," Applied Artificial Intelligence: An International Journal. Volume 13, Number 1-2, January-March 1999. pp 91-128.
- R. Simmons, S. Singh, D. Hershberger, J. Ramos, and T. Smith. "*First Results in the Coordination of Heterogeneous Robots for Large-Scale Assembly*", ISER 2000.
- M.B. Dias and A. Stentz "*A Free Market Architecture for Distributed Control of a Multirobot System"* Proceedings of the 6th International Conference on Intelligent Autonomous Systems (IAS), Venice, Italy, July, 2000.
- A. Stentz and M.B. Dias "*A Free Market Architecture for Coordinating Multiple Robots"* tech. report CMU-RI-TR-99-42, Robitics Institute, Carnegie Mellon University, December, 1999
- I. Foster, C. Kesselman, S. Tuecke. "*The Anatomy of the Grid: Enabling Scalable Virtual Organizations"*. International J. Supercomputer Applications, 15(3), 2001.
- I. Foster, C. Kesselman. ***"****Globus: A Metacomputing Infrastructure Toolkit****."*** Intl J. Supercomputer Applications, 11(2):115-128, 1997.