Microsoft Robotics Studio: A Technical Introduction

Standardizing Robotic Coordination and Control

BY JARED JACKSON

ICRA'07 Panel on the Future of Robot Operating Systems

The January 2007 cover story of *Scientific American* was "Dawn of the Age of the Robots" by Bill Gates, founder of Microsoft. Gates compares the fragmented, homebrew culture of home computers in the 1970s with today's culture around home robots and automation, which he feels is poised for a similar transformation. Gates argues that as Microsoft BASIC was a catalyst for portability, the new [(MSRS) www.microsoft.com/robotics] will facilitate progress by providing libraries for concurrency and distributed wireless control of mobile robots and a growing variety of peripheral physical devices.

To facilitate a discussion within the robotics and automation research community, we organized a panel at the IEEE International Conference on Robotics and Automation (ICRA'07), which was open to all conference participants. The panel cochairs were Ken Goldberg, vice president for Technical Activities, and Bruno Siciliano, ICRA'07 program chair. The panelists were Tandy Trower (Microsoft Robotics Group, USA), Herman Bruyninckx (Katholieke Universiteit Leuven, Belgium), Gian Paolo Gerio (Comau Robotica, Italy), Nobuto Matsuhira (Corporate R&D Center, Toshiba, Japan), and Paolo Pirjanian (Evolution Robotics, USA).

The panel began with a presentation of the MSRS by Tandy Trower, followed by presentations by each of the four other panelists, and a question-and-answer session with the audience. Tandy Trower gave a concise overview of the MSRS motivation and goals, noting that Microsoft makes the system available for free to students and nonprofit researcher groups. The other panelists provided perspectives on other efforts to develop new robot operating systems. In particular, concerns were raised about how Microsoft's proprietary MSRS might limit competition and may not yield the best possible system. Several in the audience urged Microsoft to make its system an open source.

—Ken Goldberg

Digital Object Identifier 10.1109/M-RA.2007.905745

Microsoft Robotics Studio (MSRS) was publicly released in December 2006 with the explicit goal of providing an industry software standard for robot control. To become a viable standard, several technical challenges needed to be solved. In this article, we examine the composition of MSRS, looking generally at its architecture and specifically at its solutions for concurrency, distribution, abstraction, simulation, and programmer interaction. We also examine briefly the emerging industry and academic adoption of the robotics studio.

An Overview of MSRS

To understand how MSRS interacts with robots, we must first understand what the platform understands a robot to be. Simply stated, a robot is any system of related sensors and actuators with which electronic communication is possible. To interact with these robotic systems, MSRS facilitates the mapping between decoupled software modules and hardware components or subsystems of the robot [2].

The architectural design of MSRS follows the representational state transfer pattern [4]. Interaction with a robot is achieved through the use of multiple software services, similar in nature to Web services. These services are highly decoupled, providing the ability for modular reuse of the code. A distributed messaging system is implemented for communication between services, which allows services to interact both on the same computer system, or node, and across a variety of network protocols.

The combination of the service model architecture with the generalized view of what a robot is allows developers to

1070-9932/07/$25.00©2007 IEEE

use MSRS broadly in manipulating and observing robot functionality. Services are not just limited to drive and sensor interaction but can also include implementations for Web-based error reporting, industry floor camera observation, wireless communication, etc. One example of the service to hardware mapping is illustrated against the hypothetical robot in Figure 1. In the illustration, individual software services are authored to interact with motor and sensor components as well as PC control and Bluetooth devices on both the PC and robot. In addition to interleaving, arbiters can be configured to use choice patterns to choose handlers, based on particular conditions, and can use join expressions to merge the handling of multiple ports and messages.

The interaction of services in a particular control system is defined through the use of a configuration manifest file. The manifest file is XML based and makes use of the provided functionality to uniquely identify each service available to MSRS. The manifest file tells MSRS which services to start up by defining partnerships that relate one service to another. These partnerships not only facilitate the proper initialization of the robotic control at start up but also allow for registration between services, message passing, and fault notifications.

Each service can be expressed as a state machine and is available for examination through the network. Serialization into a Simple Object Access Protocol (www.w3.org/TR/SOAP) (SOAP) based XML format is provided by MSRS, and the unique identification mechanism yields an URL by which the state data may be viewed through any Web browser. MSRS also provides service developers an interface for each service to interact with HTTP calls into the service. Thus, the developer of the service may use XSL or any other programmatic transform to present the service state in a form more rich than raw XML. Figure 2 shows the XML and the transformed HTML view of a sample service.

Several utility services are also provided by the robotics studio, most of which automatically load whenever any robotic control service is started. A control panel service provides a Web interface to the user enumerating all currently running services and providing links to each. A message logging service also runs with built-in filtering for helpful debugging views of the system. There is also a resource diagnostic service that provides additional information to assist in debugging and performance evaluation.

MSRS is implemented using .NET. To write services directly against MSRS, it is therefore necessary to use a .NET language. Typical service implementations would be written in C#, though other languages such as Visual Basic, Managed C++, and IronPython offer alternatives. It is also possible to make use of the SOAP interfaces for each service to interface with other programming platforms; however, this approach will require more development effort and will suffer performance losses in message encoding/decoding.

There are two significant limitations that result from this type of implementation. The first is that MSRS is often not an ideal platform for directly managing real time systems (RTSs) that require high frequency monitoring. Because memory

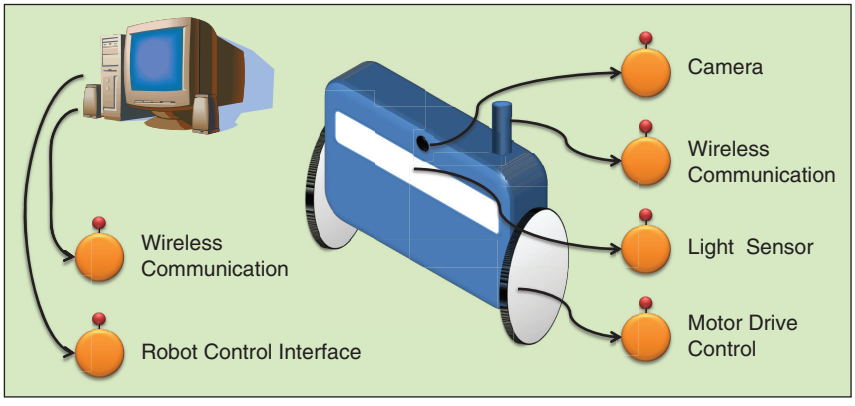

Figure 1. *Typical service configuration for a mobile robot.*

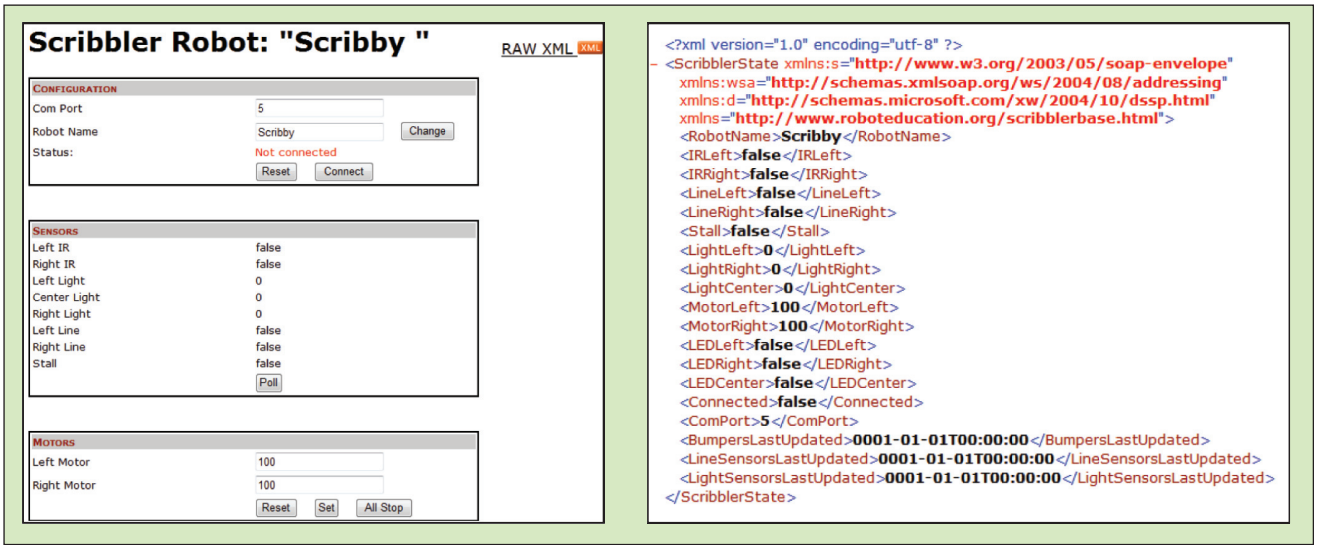

Figure 2. *Web views of an MSRS service.*

management is handled by .NET, there is no guarantee that any particular service will not be interrupted. These interruptions are typically on the order of a few milliseconds, but even that brief period can cause an unacceptable disruption to some RTSs. The solution to this problem is to keep the RTS code running separately in an unmanaged environment and then write an MSRS service that can interact with the RTS and help it communicate with the rest of the robot system.

The second limitation is that MSRS currently requires that all services run in an environment in which a full .NET library implementation is present. Many onboard robot processors do not have the ability to host such an environment. MSRS services cannot be hosted directly on these robots. In these cases, robots need to be tethered, through wired or wireless communication, to a controlling computer acting as the robot brain. In July 2007, Microsoft released a version update to MSRS, which allows running services in a .NET compact environment such as a Windows CE platform. This addition functionality greatly extends the number of supported hardware solutions but still imposes a limitation for a great number of existing robots.

With this general understanding of the architecture of MSRS and the design patterns it encourages developers to adopt, we can look a little more closely at some of the important features of the platform and the technical challenges the developers at Microsoft had to solve to implement them.

Core Features

Concurrency and Coordination

Robots by their very nature have many concurrent operations. Threading and the more recent addition of multiple core and multiple processor PCs position the computer to be a excellent controller of these highly parallelized operations.

Development and particularly debugging of software systems with many parallel processes has historically been a very difficult and costly undertaking. To address this challenge, Microsoft Research developed a .NET software library called the Coordination and Concurrency Runtime (CCR) [3]. The CCR is used extensively in MSRS for handling state updates and message processing.

Under the CCR, memory locking and communication amongst the various independently operating processes is abstracted away from the developer. Within MSRS, this abstraction is provided at the service level. While a service can generally be thought of as a state representation and associated delegate functions for handling state manipulation, the mechanisms behind those interactions are actually quite complex. Fortunately, the CCR is able to hide most of this complexity from the developer.

The workings of the CCR within MSRS services are illustrated in Figure 3. Each service defines a primary data port through which messages arrive for handling by the service. A first-in first-out (FIFO) queue is associated with each port, as is an arbiter class instantiation. The arbiter is configured by the programmer to know which delegate handlers can react to a particular message. This configuration also includes specification as to which delegates can safely run in parallel and which are mutually exclusive. Beyond the handler definition and arbiter configuration, all memory locking and process assignment is handled automatically by the built-in CCR.

Distributed Messaging

To address the highly decoupled nature of the service architecture of MSRS, an efficient message passing technology was integrated into the platform. MSRS makes use of a second software library developed within Microsoft Research called the Decentralized Software Services Protocol (DSSP) [5].

It is the DSSP system that launches services from its manifest descriptions, provides for partnering, and facilitates communications between message ports on individual services. DSSP is designed to scale message encoding appropriately, with closely connected services requiring little packaging to services distributed across networks that require much more.

In the most highly distributed cases, services message each other using standard SOAP protocols. In addition to this model of communication, MSRS also optimizes to two other less distributed cases. In the first case, multiple services exist on the same processing unit or node. Messaging between nodes in this case does not need the overhead of SOAP packaging and traversal of the entire TCP/IP stack. In the second, even more tightly knit,

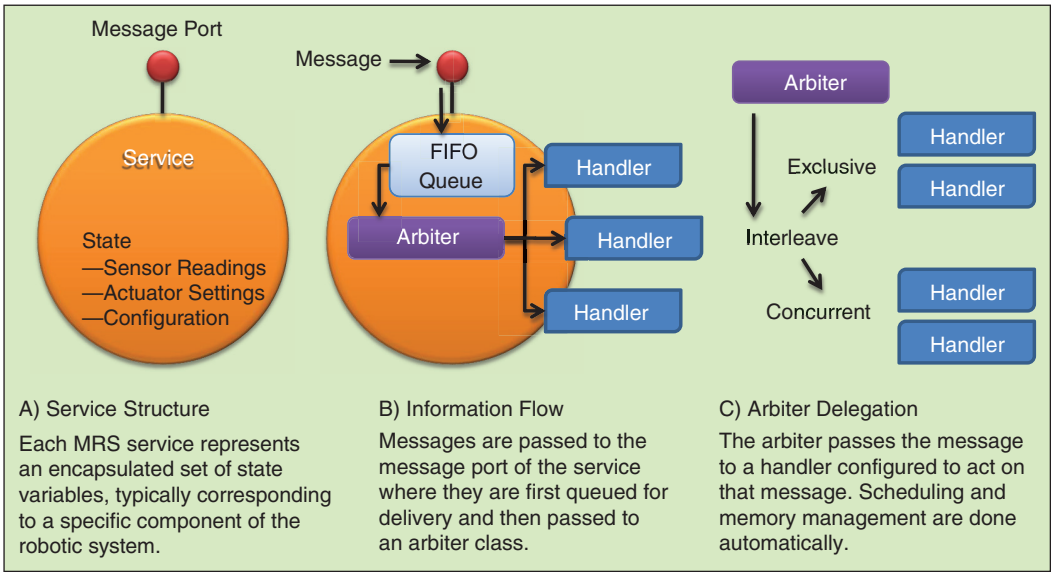

Figure 3. *Service architecture and concurrent message handling.*

case, multiple services are run in the same process and memory context. In both cases, the robotics studio passes information between services with the minimum amount of overhead and repackaging of data transmissions. The result of this flexible design is much quicker communication than traditional methods, thus opening the software platform to broader applications. Figure 4 illustrates how the encoding and message passing is performed according to differing degrees of connectivity, with benchmark frequency results for simple message passing.

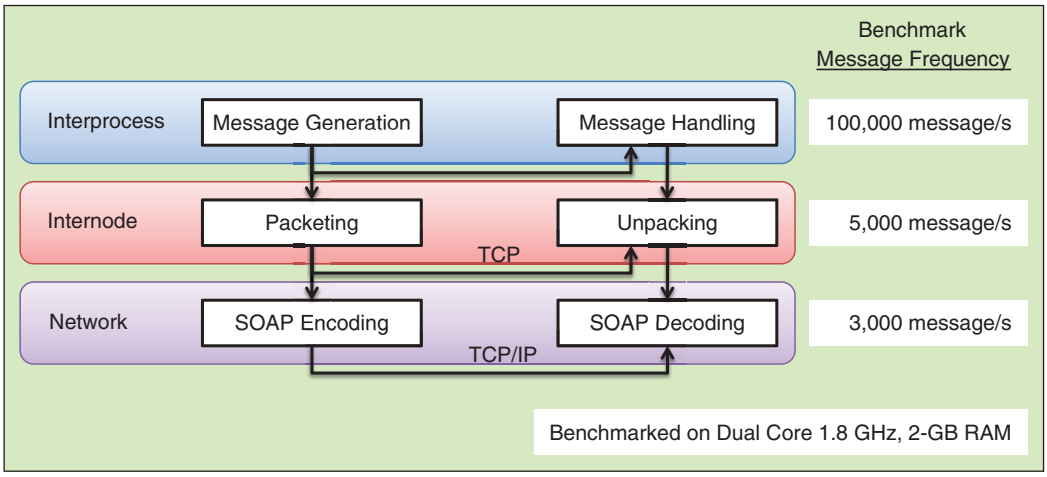

Figure 4. *Distributed messaging.*

Abstraction

One challenge accompanying the use of a decoupled service architecture is that the developer is prevented from being able to use an inheritance mechanism to interact with services that perform a similar function. In early design of the robotics studio, an attempt was made to define a hierarchy of robot components. When applying this hierarchy to existing robots, almost invariably some unique aspect of the robot would require yet another addition to be added to the design. The end result was bloated and complex, and the whole approach was abandoned.

The abstraction mechanism now provided in the MSRS uses a mechanism similar to the interface construct found in object oriented programming. Generic contracts are defined by MSRS for common elements of a robotic system. Developers of new services can specify that the service conforms to one or more of these contracts. Other services can then use a detection mechanism to discover all currently running services that conform to a particular contract. Once discovered, these services can be messaged without knowing the specifics of the hardware behind that service.

There are many contracts that come included with MSRS. Each of these can be found in a software library called RoboticsCommon.dll. A partial list is illustrated in the Sidebar.

To illustrate the use of contracts with an example, consider the set of services shown in Figure 5. In the illustration, an application with a graphical interface is written and wrapped into an MSRS service. The GUI control service uses the built-in detection mechanism to discover if any drive and/or Webcam services are currently available on a particular node (local or networked). Discoverable services could include a Lego NXT drive controller, which has no camera, a simulated NXT with a simulated camera built in, an iRobot Create with a webcam dongle, or any number of other robot configurations.

Services that implement contracts are not limited to the functions provided by those contracts and can react to any message the service developer sees fit to handle. Thus, features unique to a particular robot can easily coexist in a service implemented according to a contract. Accessing unique features will require the partnered service to know of those features, while services written to be applied generically can still interact with the same service.

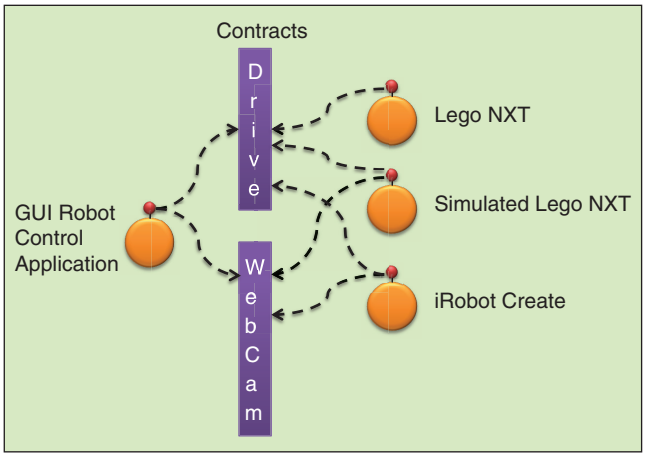

Figure 5. *Example of contract use.*

Generic Contracts

- Analog Sensor
- Articulated Arm
- Battery
- Camera
- Contact Sensor
- Drive
- Motor
- Sonor
- WebCam

Simulation

A powerful three-dimensional simulator is also included in MSRS. Unsurprisingly, Microsoft's DirectX technology is used to provide the graphics framework. In the simulator, services can define their own entities. These entities make up the objects placed in the simulated world. Entities are not limited to the robots themselves but can include environment objects like obstacles, walls, floors, etc. Each entity is provided with a set of

callback functions for dealing with events in the simulated world, such as collisions and frame updates.

The simulator in MSRS not only provides a graphical simulation but also integrates a powerful physics simulator. Physics simulation is performed using a physics engine developed outside of Microsoft by Ageia. Entities in the simulated world specify a physical description complete with friction coefficients, mass, and center of gravity. Simulated contact takes into account force, torque, momentum, and resistance when updating the positions of each frame. The screen shots in Figure 6 show a view of a simulated world, showing both at its graphical and physics representation.

Both the graphics and physics simulation can be performed through software on Windows XP and higher. In addition, graphics cards supporting DirectX 9 will accelerate rendering. Ageia's physics engine also allows hardware acceleration through the optional integration of the physics processing unit.

Programmer Interaction

As mentioned in the overview, MSRS services are authored using any .NET compatible language. A series of tutorials, including the installation of the platform, introduce both basic and advanced uses of the robotics studio and show implementations of services on a variety of commonly available robots. Elements of the studio make use of built-in .NET features such as attributes on the service classes and methods, yielding iterations, and generics. The tutorials include examples of MSRS services running in C#, Visual Basic, and IronPython—the .NET compatible version of the Python language.

In addition to standard .NET languages, MSRS also includes a visual programming language (VPL) of its own design. The VPL is designed both for hobbyists with little understanding of programming and advanced users looking to rapidly prototype designs. VPL programs are translated into C# code before being compiled and run. The generated C# code is available to the developer to make iterative changes. Currently, MSRS does not support round tripping code edits from the VPL to a standard Integrated Development Environment (IDE) and back. Figure 7 shows a simple service written with the VPL.

Interacting with running services is not limited to code written against the .NET environment. Each service provides a mechanism for interaction through the Web. Interacting with

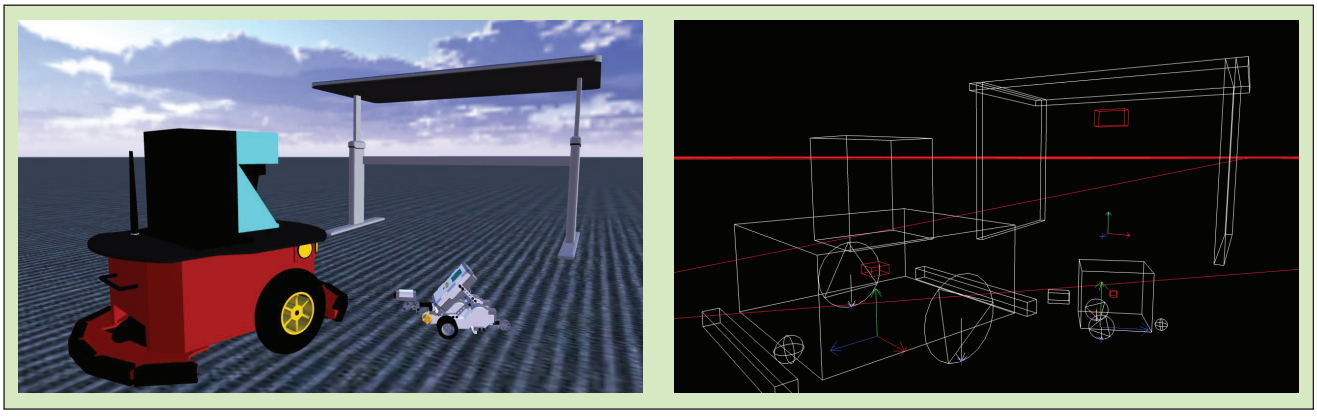

Figure 6. *Simulation screen shots.*

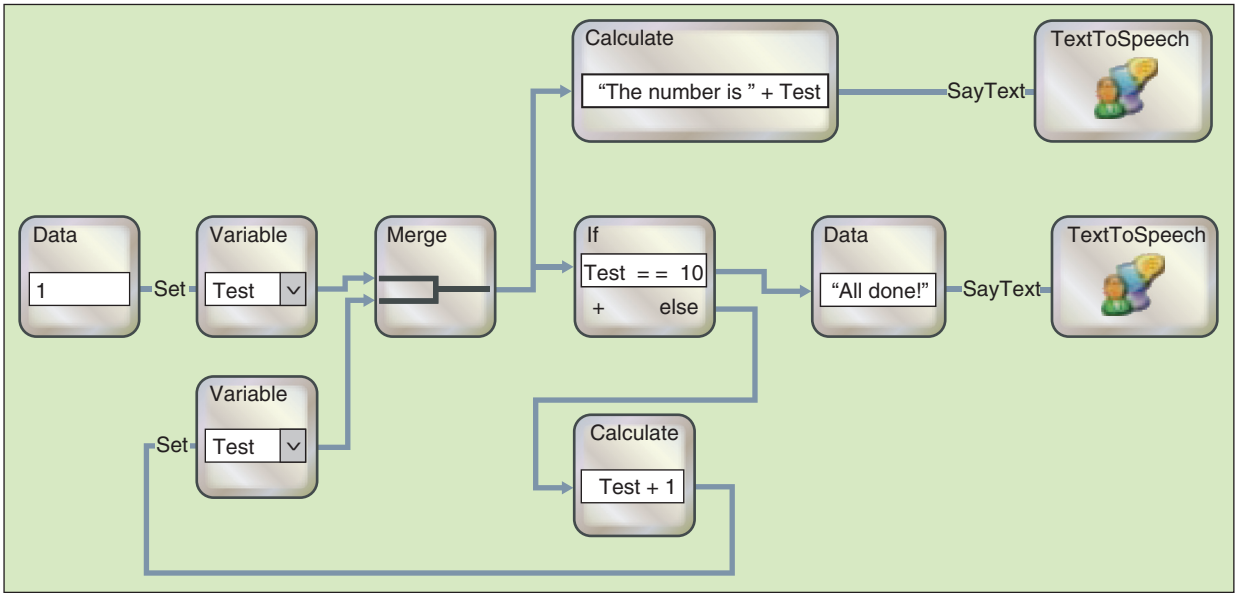

Figure 7. *Programming with the visual programming language.*

a service can be as simple as typing in the correct URL into a Web browser. Scripting languages such as JavaScript and any other language with an adequate Web service or networking library can also be used to interact with MSRS services.

Adoption of MSRS

Becoming a standard for robotic development is not merely a technical challenge. Partnerships within the industry and academia are needed to prove Microsoft's claim that their technology can finally unite the fractious software development situation currently existing in the robotics community. To this end, several partnerships have been made within the community. In addition, MSRS is currently licensed for free download and use to anyone using it for noncommercial purposes.

Industry Partners

Included in the MSRS are several services for robots common to the marketplace. These services include control for robots from FischerTechnik, iRobot, Kondo, Lego, Mobile Robots, Parallax, and Traxster. Other partners, too numerous to list here, are working with Microsoft to provide solutions to their robots or robotic components. A quick visit to the MSRS Web site will provide a list of these companies.

The Institute for Personal Robotics in Education

In academia too, several efforts are being undertaken that make use of MSRS technology to further robotics and technology education.

One such program aiming to better technology education is the Institute for Personal Robotics in Education [(IPRE) www.roboteducation.org]. The IPRE is a joint effort between Georgia Technical University, Bryn Mawr College, and Microsoft Research aimed at improving computer science education. The initial goal of the IPRE is to introduce robots into early computer science education, getting students excited about technology and the field of computing [1].

The Institute is only in its first year but has ambitious goals of disseminating course material and other offerings to hundreds of colleges and universities. MSRS's capabilities of adapting to differing sets of hardware and software tools are essential to bringing these goals to pass. The simulation, visual programming language, and Web-based communication features all open the field of robotics to students just entering or evaluating this exciting and sometimes demanding field of study.

Center for Innovative Robotics

At Carnegie Mellon University, another robotics institute uses MSRS to further the collaboration between robot manufacturers, researchers, and enthusiasts. The Center for Innovative Robotics (www.cir.ri.cmu.edu) provides a central repository for robotic knowledge and discussion. As it grows, the center's knowledge base will broadly span the field of robotics and aims to speed up innovation through the use of shared tools and innovation.

Conclusions

MSRS provides a broad suite of technological solutions to problems common in robotic development. The combined system of concurrency control with efficient distributed message passing will cut hours out of the development time on robotic systems. Simulation and visual programming can assist with rapid prototyping, and broad acceptance by the robotics community will enable more sharing of solutions. Limitations of the studio are still evident in areas such as real time computing and integration with low-level processors. Still hybrid solutions are available that overcome these limitations.

MSRS understands a robot to be any system of related sensors and actuators with which electronic communication is possible.

MSRS is a new entrant in the quickly growing field of robotics. The growth of the robotics market means robots are now poised to bring greater utility to households across the world. New ideas and broader acceptance come every day, and in order for the market to expand in the same way as other successful technologies, such as the personal computer, platforms that ease the process of innovation in robotics will be crucial. While it still remains to be seen if MSRS will be accepted by the industry as a gold standard, it is clear that it has already turned that which is complex into a simple starting point for a future of innovation.

Keywords

Robotics, development, visual-programming, concurrency, distribution.

References

[1] D. Blank, "Robots make computer science personal," *Commun. ACM*, 2006.

[2] G. Chrysanthakopoulos and H. Nielsen, (2007). *Microsoft Robotic Studio Runtime Architectural Overview.* [Online]. Available: http://www.microsoft.com/winme/0703/29490/Architecture_Overview/Local/index.html.

[3] G. Chrysanthakopoulos and S. Singh, "An asynchronous messaging library for C#," in *Proc. Annu. Conf. Object-Oriented Programming, Systems, Languages and Applications*, Anaheim, CA, 2003.

[4] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, Univ. California, Irvine, 2000.

[5] H. Neilsen and G. Chrysanthakopoulos. (2006). *Decentralized Sofware Services Protocol: DSSP.* [Online]. Available: http://msdn.microsoft.com/robotics/media/dssp.pdf.

Jared Jackson is a research software development engineer in the Extended Research and Programs group within Microsoft Research. He is currently contributing to the Institute for Personal Robotics in Education, a partnership between Georgia Tech, Bryn Mawr College, and Microsoft Research. He is a graduate of Harvey Mudd College and Stanford University.

Address for Correspondence: Jared Jackson, One Microsoft Way, Redmond, WA 98052, USA. E-mail: Jared.Jackson@microsoft.com.