Remote control of a standard ABB robot system in real time using the Robot Application Protocol (RAP)

Per Cederberg **Magnus Olsson**
Gunnar Bolmsjö
{pcederberg, molsson, gbolmsjo}@robotics.lu.se
Div. of Robotics, Lund University, P.O. Box 118, S-221 00 Lund, SWEDEN

ABSTRACT

Sensor equipped robots should benefit from using a world model when process-dependent decisions such as collision control have to be taken in real-time. The world model can quite easily be represented and updated in a CAR (Computer Aided Robotics) application. A missing link, however, is the ability to remotely control the robot motion during execution. This paper shows that the Robot Application Protocol, RAP, which provides an interface to standard ABB S4 robot controllers, can be utilized to let a remote program control the robot in 10 Hz, i.e. to create motion on the fly in joint space by interacting with the running RAPID program on the robot controller. The results are currently used in ongoing welding experiments focused on process control.

Keywords: robotics, remote control, real-time, sensor, simulation, CAR

1 INTRODUCTION

The current trend towards fast product changes along with customization and optimized design using new materials and manufacturing processes put greater demand on manufacturing operations with respect to control performance and the resulting productivity and quality. Considering the number of robots used in industrial automation, the use of advanced sensors is still small. One area where sensors are important is arc welding where products based on new materials decrease the overall dimension (plate thickness) and increase the general need for keeping tight tolerances during welding. The use of industrial robots in this context generally requires an integrated approach where product data defined within a CAD (Computer Aided Design) environment is taken as input and applied within a CAR software that enables modeling, simulation and programming of robot operations.

The Robot Application Protocol, RAP, provided by ABB is an interface to the S4 robot controllers. Four major groups of services are covered: general management, variable access, file management and program control. Despite a few cavities, including sparse documentation, the RPC (Remote Procedure Call) based Robot Application Protocol allows a remote program to control the robot in real-time (although with limited bandwidth), i.e. to create motion on the fly by interacting with the running RAPID program on the robot controller.

Important work in this area is presented in [1] where RAP is used to implement remote access to the robot controller, including programming and monitoring in a Windows-centered environment. Application examples using the proposed solution are also briefly presented. A different approach is presented in [2] where a Matlab toolbox has been created which interfaces RAP and demonstrates the setup with an ABB robot equipped with a force/torque sensor.

This paper gives a brief overview of major RAP functionality, limitations and applicability to situations where robot motion cannot be limited to preprogrammed motions in a RAPID program. An operating system independent method to access a running RAPID program is described. It differs somewhat from the strategy used in [1, 2]. Especially sensor guided robot systems should benefit from using RAP. We have previously shown how to guide an ABB robot which allowed remote control in real-time and in joint space [3, 4, 5, 6, 7]. In the experiment, Envision, a sophisticated CAR application from Delmia, controlled an ABB IRB2000 robot with an attached ServoRobot M-Spot laser scanner. The system performed a straight fillet joint weld on a workpiece of which position and orientation differed from the original nominal pose. In ongoing research, similar experiments are planned, now utilizing an unmodified ABB IRB2400/16, RAP and the communication tools described in this paper. As a positive side effect of the research, a “compiler tool”, based on RAP and developed during the evaluation process, is described. The tool lets the user edit, compile and run RAPID programs from a remote workstation using *emacs* (a standard UNIX editor) and the developed error formating can easily be adjusted to work together with other editors.

2 MATERIALS AND METHODS

The experimental setup consists of a master that controls a virtual and a real slave robot. The master cre-

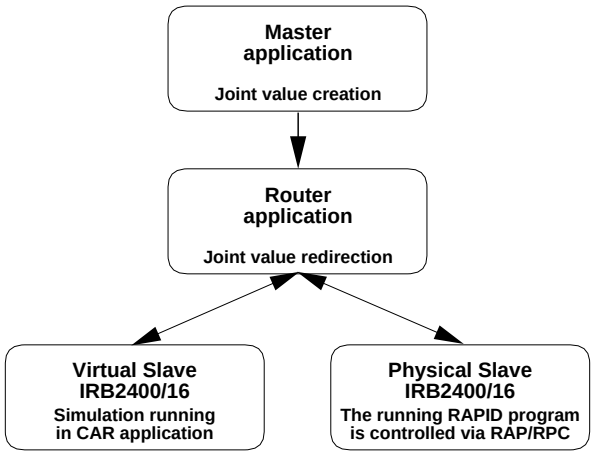

Figure 1: *Experimental setup. Joint values created by the master are routed to either the physical or virtual slave. RAP/RPC-based messaging are utilized between router and the running RAPID program on the physical robot controller. The double directed arrow shows the handshaking process between the slave program and the router.*

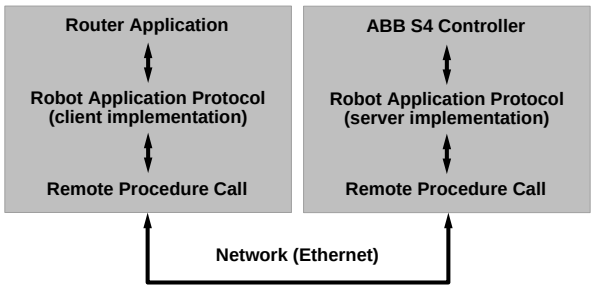

Figure 2: *The different levels of APIs when the router is invoked by the master application. RAP uses TCP/IP as transport protocol.*

ates joint values by interpolating between pre-defined tag-points. Two different master applications have been tested, Envision, a CAR application from Delmia in which three tag-points in a triangle defined the task, and a simple application that read pre-created joint values from a file, values that earlier was created by moving along the triangle shaped path in the CAR application. These values are sent to the slave robot through a router application. The router re-direct the joint values to either the physical or virtual robot. RAP/RPC-based messaging are utilized between router and the running RAPID program on the physical robot controller, see Figure 1.

The router is based on RAP which uses Sun's RPC and the External Data Representation (XDR) protocol to transfer requests and replies to/from the remote robot controller where the RAPID language is executed. The different communication levels are shown in Figure 2 and are explained below.

2.1 RAPID language

The program consists of a number of instructions which describe the work of the robot in a Pascal like syntax. There are specific instructions for the various commands, such as to move the robot or to set an output, etc. There are three types of routines: procedures, functions and trap routines and three kinds of data: constants, variables and persistents. Persistents are variables that can be reached from the outside world. Other features in the language are: routine parameters, arithmetic and logical expressions, automatic error handling, modular programs and multitasking [8].

2.2 Remote Procedure Call and External Data Representation

Remote procedure calls are a high-level communication paradigm that allows programmers to write client/server network applications using procedure calls that hide the details of the underlying network. The RPC model is similar to the local procedure call model where the caller places arguments to a procedure in a well-specified location (such as a result register) and transfers control to the procedure. When the caller eventually regains control, it extracts the results of the procedure from the location and continues execution [9]. RPC uses XDR to establish uniform representations for data types in order to transfer message data between machines [10]. Sun's RPC and XDR are freely available on numerous platforms.

2.3 Robot Application Protocol (RAP)

The Robot Application Protocol provides a set of services that makes it possible to monitor and control the robot from an external computer. These are grouped into four classes: general management, variable access, file management and program control services. The general management services are support services for all other services, e.g. open and close a connection to a specified server and restart of the controller. RAP is using named variable objects to get information from the robot-system or affect the robot system, e.g. to read and write RAPID defined and predefined system variables and event handling. An event in the system can be subscribed for and as a result of that subscription, a spontaneous message will asynchronously be sent to the external computer when the event occurs. RAP file management provides the functionality to access files on the memory devices in the robot system, e.g. to open, read, write, close, rename and delete a file [11, 12, 13].

2.4 High-level remote motion control

Normally, a RAPID program needs no invocation from the outside world after execution has been initiated. It is only possible to send data with RAP (as oppose to instructions; a limitation that has been circumvented in [1] by introducing a switch statement in the RAPID program where each selector defines a predefined and

possibly complex service). A special RAPID program with a designated sequence of move instructions has to be down-loaded to the controller. The RAPID program needs to be carefully designed to be able to let a remote application control the robot's motion in a master-slave fashion. The relevant part of the program consists of a loop where a set of move instructions continuously are executed as shown below.

```
!RAPID program executing on
!robot controller
...
PERS num           pnum  := -1;
PERS num           p0set :=  0;
...
PERS robtarget     p0:=[...];
...
WHILE NOT aborted DO
  WaitUntil p1set <> 0;
  p1set := 0;
  pnum := 0;
  MoveL p0, v, z, tool0;

  WaitUntil p2set <> 0;
  p2set := 0;
  pnum := 1;
  MoveL p1, v, z, tool0;

  WaitUntil p3set <> 0;
  p3set := 0;
  pnum := 2;
  MoveL p2, v, z, tool0;

  WaitUntil p0set <> 0;
  p0set := 0;
  pnum := 3;
  MoveL p3, v, z, tool0;
ENDWHILE
...
```

The loop represents a circular buffer of n *robtarget* structures $p_0 \cdots p_{n-1}$ which contains position, orientation of tool center point (tcp) and configuration of robot and external axes. The variables are declared as *persistent*. During execution, these structures are dynamically set (with some latency) with joint values provided by the master. The router keeps track of which move instruction that is to be executed by the robot controller, i.e. which robtarget structure to update at a specific time.

The *MoveL* directive refers to a via movement and v and z denotes tcp speed and the *zonedata* structure respectively. Zonedata is used to specify how a position is to be terminated, i.e. how close to the programmed position the axes must be before moving towards the next position. For instance, at some point of time during the move from $p1$ to $p2$, both $p2$ and $p3$ must be known to the robot control system. To be able to prepare for the next movement a fourth point is needed.

Even though a RAP call returns synchronously, there is no guarantee that the robtarget structure in RAPID is updated when *writeRobTarget()* returns. Since RAP calls that write data to RAPID variables in practice are asynchronous, there is need for a mechanism to be certain that a particular variable holds the data previously written to it. This handshaking problem has been resolved by using busy wait (a polling method) in the RAPID code as well as in the router. The $pnum$ variable in the running RAPID program is continuously monitored and when it eventually becomes updated in the RAPID program, the next robtarget structure update is sent from the router to the robot controller.

```
/* Router pseudo code to handle */
/* routing of data and handshaking */
/* between master and slave */

static int pnum = -1;

int pnumChanged()
{
  RAPVAR_DATA_TYPE data;

  readRAPIDVar("pnum", data);
  if (data.RAPVAR_DATA_TYPE_u.num != pNum) {
    pNum = data.RAPVAR_DATA_TYPE_u.num;
    return 1;
  }
  return 0;
}

void routeRobTarget(ROBTARGET robTarget)
{
  /* set two pnum's ahead, i.e. */
  /* if pnum = 0, set p2set = 1 */
  /* in RAPID program */
  int pXset = (pnum + 2) MODULUS 4;
  writeRobTarget(robTarget, pXset);
  writepXset(pXset);
}

void routeJoints()
{
  ROBTARGET   robTarget;
  JOINTVALUES joints;
  do {
    if (pnumChanged()) {
      joints = sendJointRequestToMaster();
      robTarget = kinematicCalculation(joints);
      routeRobTarget(robTarget);
    }
  } while (!aborted);
}

void main()
{
  if (compileProgram() == SUCCESS) {
    connectMaster();
    /* initialize first two robtargets */
    initializeRobTargets();
    startRAPIDExecution();
    routeJoints();
  }
  else
    displayErrors();
}
```

To avoid having the speed of the master exceeding the speed of the slave, the master only sends joint values after receiving a request message from the router. If the

master is unable to send values upon a request, the slave robot will come to a temporary stop until the master is ready to resume delivery of joint values.

2.5 The router as a RAP application example

Some of the strengths of RAP are shown in the supporting parts of the router. Besides handling the real-time issues of routing values from master to slave, the router provides a RAPID compile environment within *emacs*, a well known LISP based editor. By issuing a compile command in emacs, the RAPID program will get syntax checked and any errors will be shown with row, column and error message in a second window. By clicking on the error, the cursor will mark the offending line in the RAPID program, see Figure 3.

Behind the scene, the actual compilation is performed remotely on the robot system. By using RAP, the RAPID program is sent to the robot controller and is loaded and checked. The errors are saved in a log file on the robot controller and are transferred back to the router and displayed to the user in an user-friendly fashion. If the RAPID program passes the syntax check, the user may choose to automatically run it.

2.6 Using a virtual slave robot

When a virtual robot is used as slave it is modeled in a CAR session and the same API (Application Programming Interface) as for the physical slave is used. To mimic the speed deficiencies caused by RAP-RAPID interaction in the real system, latencies and speed limits can be specified.

3 RESULTS

It is possible to control a standard robot system on-line from a remote workstation using RAP, RPC and a carefully designed rapid program and handshaking routines. A transfer limit of approximately 10 Hz has been measured using a local Ethernet-based network. An environment for remote editing and compilation of RAPID programs is provided as an example of benefits of RAP utilization.

4 DISCUSSION

RAP is an add-on to the robot control system that has been designed for monitoring purposes more than heavy interaction with the RAPID program. As mentioned before, when a robtarget structure is sent from the router to affect a variable in the RAPID program, the RAP call responds (with no error) at some time before the variable in the RAPID program is updated and useful in a move instruction. In practice, neither RAP nor RAPID give any choice but to use busy wait (a polling method) to synchronize the router and the RAPID program. Based on these prerequisites, a transfer limit of approximately 10 Hz has been measured using our local network. The *SCWrite* RAPID instruction can be used to send both single and array variables from the RAPID program to the client. To receive the values, the client has to act as a RPC server, either as a thread or as a separate application. However, since the instruction itself is asynchronous, it is not suited for handshaking purposes.

RAPID executes in a low prioritized thread on the ABB robot controller. Latencies measured as time between data request and response varies dependent of the task the robot currently handles. Latencies has not been specifically measured in this paper but [1] reports 15 to 20 ms on average.

It is for some reason not possible to send jointtarget structure values directly using RAP even though the structure is defined in the RAPID language. As a work around, a kinematic model of the robot and some simple calculations to represent the rotation with quarternions, easily create a robtarget structure from joint values. A second method could be to define a jointtarget structure in RAPID where the different joint values are simple floating point variables, and make a RAP call for each of the individual variables. A third method might be to define an array of floats in RAPID and use the array indices as the joint values in the jointtarget structure. This would be advantageous to the second method in that only one RAP call would be needed to send all six array values. In this paper the first method has been used in order to minimize the number of RAP calls and to add as little overhead as possible to the RAP server and to the RAPID interpreter.

Because of the precise timing needed in the handshaking process, the choice of master becomes important. The master has to be able to provide new joint values when asked for by the router. Since a CAR application normally defines simulated time, it has turned out to be difficult to correlate simulated time with “real” time provided by the robot controller. As a result, the physical robot’s lag behind the virtual increased during motion. The problem was eliminated using the simple master which provided joint values timely. Needless to say, the difference between the router and the master is purely conceptual; they could of course be built as one application.

Even if motion has been in focus in this paper, RAP offers a wide range of other services such as reading and writing of several variables using only one RAP call, advanced file management services and device access. Some services are missing, however. As mentioned before, the RAPID variables of jointtarget structure type cannot be read or written without a workaround. Also mentioned, the asynchronous RAPID function *SCWrite* cannot be used for effective handshaking purposes. Furthermore, [12] states that the function might fail if the *SCWrite* messages comes “so close that they cannot be sent to the external computer”, a characteristic that does not relate well with performance. A more reliable write function would be helpful. A third desirable function would be an atomic test and set. Then the two lines

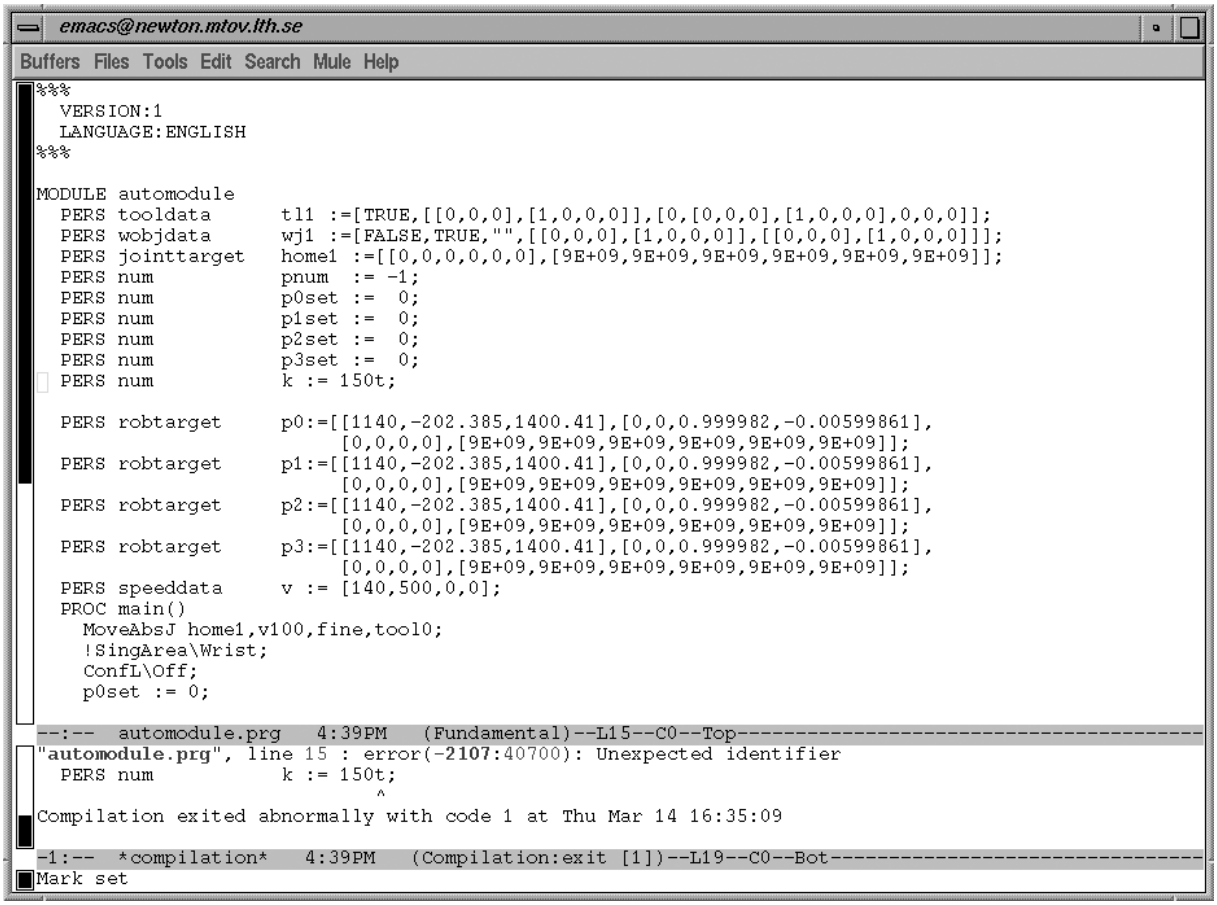

Figure 3: *The router provides an emacs compile environment which allows the user to remotely edit and execute RAPID programs (the program in the figure is just provided as an example, please disregard from details). After an unsuccessful compile, the error is shown in a second window. By clicking on the error, the cursor will mark the offending line in the RAPID program.*

```
...
WaitUntil p1set <> 0;
p1set := 0;
...
```

could be exchanged to one safe function where a single RAPID operation reads, tests and writes a variable.

In the ongoing experiments with a sensor guided robot, a master has been developed which includes parts of the virtual model needed for motion. This simplified nominal world model is exported from the CAR application before execution. During execution, other applications may subscribe to dynamic changes to the virtual model caused by sensor information. A subscribing CAR application should for instance be able to use its sophisticated world model to monitor changes, perform collision checks and propose error recovery strategies.

5 CONCLUSIONS

By using the RAP based router and a carefully designed RAPID program, it has been proven possible to, in real-time and from a remote computer, control motion in a standard robot system. The update rate of the RAPID program via RAP has been measured to 10 Hz. The solution should be of value in situations when it is desired to control the robot without having to rely on pre-programmed motion or actions such as for instance when non-trivial sensors are used. RAP has also proved to allow remote editing, compilation and loading of robot programs.

ACKNOWLEDGMENTS

The authors would like express our gratitude to our colleagues Mr. Mathias Haage and Dr. Klas Nilsson, Department of Computer Science, Dr. Anders Robertsson, Department of Automatic Control, all at Lund University and to Dr. Torgny Brogårdh and Mr. Valter Macovac at ABB Robotics, Västerås, Sweden.

REFERENCES

[1] J. N. Pires and J. M. G. S. da Costa, "Object-oriented and distributed approach for programming robotic manufacturing cells," *IFAC Journal Robotics and Computer Integrated Manufacturing*, no. 16, pp. 29–42, 2000.

[2] J. N. Pires, "Interfacing robotic and automation equipment with Matlab," *IEEE Automation and Robotics Magazine*, September 2000.

[3] P. Cederberg *et al.*, "Virtual triangulation sensor development, behavior simulation and CAR integration applied to robotic arc-welding," *Submitted to Journal of Intelligent & Robotic Systems 2001. Accepted, waiting for publication*, 2001.

[4] M. Olsson, *Simulation and execution of autonomous robot systems*. PhD thesis, Division of Robotics, Department of Mechanical Engineering, Lund University, 2002.

[5] K. Nilsson, *Industrial Robot Programming*. Lund, Sweden: Dept. of Automatic Control, Lund University, 1996. Ph.D. Thesis.

[6] "QuickTime movie of virtual system, see our website http://www.robotics.lu.se," Division of Robotics, Lund University, 2001.

[7] "The thinking machine, part 3 (video recording)," in *Autonomous Industrial Robotics. Demonstration performed by Magnus Olsson. Also published and narrated in English on http://www.robotics.lu.se*, NUTEK, 1999.

[8] "ABB Rapid Reference Version 3.2, RAPID Summary," ABB Flexible Automation.

[9] "IRIX Network Programming Guide," SGI.

[10] "AIX Version 4.3 Communications Programming Concepts," IBM, 1997.

[11] "ABB RAP Protocol Specification 1.05," ABB Flexible Automation.

[12] "ABB RAP Service Specification 1.05," ABB Flexible Automation.

[13] "ABB Ethernet Services 3.0," ABB Flexible Automation.