

AS-116.140 XML Seminar Autumn 2000

Potential Use of XML in Robotics

Sami Salmi

1. INTRODUCTION

In the last few years the amount of both industrial and special robots has risen, (and it still does). Also a new market has been arisen for sophisticated, relatively low-cost, commercial robots.

Many small research groups that don't have the expertise, facilities, or budgets to build their own robots can enter the field of robotics by buying commercially ones. As a result, there are now more laboratories capable of robotics research than there were just a few years ago. However, this has generated a problem; each vendor has developed its own software architecture, and no two systems work with one another. This has created a robotics Tower of Babel.

On the industrial robot side similar problem is on the programming and controlling languages.

Then again no single architecture could accommodate the needs of all researchers and/or industrial robot developers and programmers. For example, one researcher might need a formal planning system. Another might need a learning system.

Instead of creating an architecture, which incorporates a researcher's specific control paradigm (such as neural networks or fuzzy logic), one could build an infrastructure from which these "architectures" could hang.

2. STANDARDS FOR ROBOT PROGRAMMING

Over the years there has been projects to build up an 'Universal' programming language, but all of them has been forgotten or the robots has just evolved away from these languages.

The main effort has been on the industrial robot side and here are some examples from the near past:

- **IRL** - Industrial Robot Language. A high-level language for programming industrial robots. "IRL, Industrial Robot Language", *DIN 66312*, Beuth- Verlag 1992.
Close cooperation between robot manufacturers, robot users, and research institutes gave birth to a new high level programming language for robots. The national German standardization organization (DIN) published this language at the beginning of 1990. The language is independent of any particular robot controller or robot kinematics
- **RobotScript™** , Robotic Workspace Technologies (RWT) , Inc. Florida USA. 2000.
RWT claims that the RoboScript is the first universal industrial robot programming language. It is based on the Microsoft interpreted computer language Visual Basic Scripting Edition, or VBScript. The commands are English-like. This language can be used only with RWTs Robot Control Solution (RCS interprets the RobotScript language to the robot controller), or tested with simulation programs like Deneb's IGRIP.

Also one reference to ISO standart was found:

Almgren Roland: *Comments on Draft Proposals from the ISO on Standard Robot Programming Language*. LiTH-IKP-I-149.

But this was the only this information could be found on the web

3. XML IN ROBOTICS

At the moment the WorldWideWeb consortium (W3C) is defining a new 'language' to rationalize the web. Simultaneously other IT areas has noticed that here could be the solution for the 'Universal language', or at least a common way to present data in web, data communication, data banks and in many more applications. This new 'language' is called XML, Extensible Markup Language.

XML is a description language that allows the representation of structure and data by the use of novel tags. It resembles a lot database programming or some publication programs like latex.

XML provides good platform to share code with other research groups, but it also allows a very good possibility to document these program codes automatically and the reader can choose the document outlook or the data needed.

At the moment, to mine and Dr. Douglas Blanks knowledge only his laboratory *Artificial Intelligence and Robotics Laboratory, University of Arkansas*, and groups closely related to that laboratory are using XML based language, XRCL, in robotics.

A new program called IREDES to develop and standardize control and communication language inside the mining industry, from the organization level to the control of a mining machine (Ilkka Kauppis presentation in this seminar)

4. XRCL - THE EXTENSIBLE ROBOT CONTROL LANGUAGE

XRCL (pronounced zircl) is a relatively simple, modern language and environment designed to allow robotics researchers to share ideas by sharing code. It is an open sources project, protected by the GNU Copyleft. That means that any projects built on top of XRCL must be given back to the robotics community. This also means that you have complete freedom to the source code, and can alter it in any manner you wish.

XRCL is actually two things: a proposed language specification, and an environment for interpreting that language

XML has been chosen as a starting place for the XRCL project for three main reasons as follows:

- it has emerged as a standard method of representing structure
- it provides many support tools (e.g., parsers, editors)

- it has become popular and well-known to programmers and non-programmers alike

The vision of the XRCL language is a cross between XML and C++, but you really don't need to know either to start using the system. C++ has been picked to be the programming language; however, it would be easy to incorporate other languages, such as Java or Lisp. The system is designed to be easy to use for the novice roboticists, but extendible for the advanced researcher. An example can be seen on fig 2.

XRCL logic is based on Kurt Konolige's Saphira system. It is a fuzzy logic, behavior based robot control system.

(The basic idea of the Saphira architecture is the concepts of coordination of behavior, coherence of modeling, and communication with other agents.)

4.1 The Plan

The hope is for XRCL to support multiple robots simultaneously from various vendors. So far, a working prototype for two simulated robots, and three hardware platforms is built: the B21R, the Pioneer (fig 1.), and the HandyBoard.

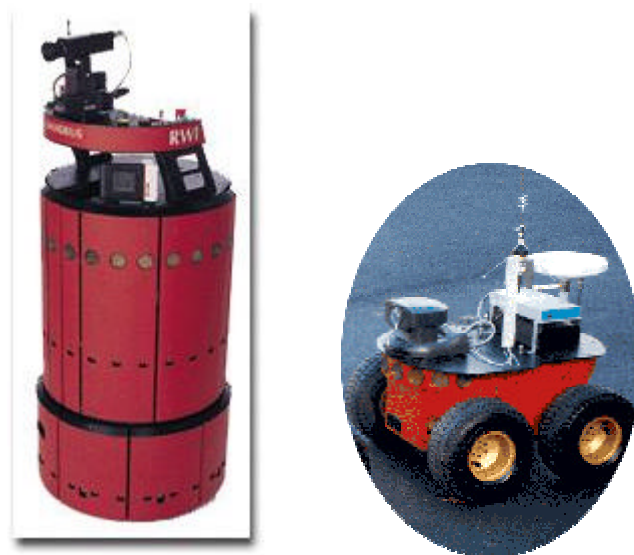


Figure 1.

a) B21R

b) Pioneer

The starting references were for two categories of developers:

Behavior developers

Developers who want to work on creating behaviors: programs written in XRCL that control the robot.

System developers

Developers who want to work on the XRCL system, they are writing the tools used by the behavior developers.

Using RWI's (*iRobot's Real World Interface Division*) CORBA interface gives the possibility of controlling any robot (or device) on the Internet. In this manner, a server runs on the robot and provides an interface to a particular piece of hardware. Unfortunately, requiring a server running for every piece of hardware, especially when you are running your controller on the same computer, is overly, and unnecessarily, complicated.

It is propose three separate projects with which to create a productive community for robotics researchers:

1. A meta-language description language, like XRCL, to describe interfaces to software and hardware components
2. An environment which is capable of running these programs
3. A standard for accessing hardware components via both the Internet (such as CORBA over TCP/IP), or directly.

4.2 Design of the XRCL

The XRCL system is a hybrid system consisting of reactive fuzzy logic "behaviors" on one end, and global "finite state machines" on the other.

On the low-level end, you will write fuzzy rules, such as:

IF too_close_to_wall THEN Speed STOP;
IF ! too_close_to_wall THEN Speed FAST;

These fuzzy rules are packaged up in little nuggets called "behaviors". Behaviors are nothing more than a collection of fuzzy rules and supporting calculations, like above.

On the other end, we have what is called "finite state machines" (or FSM) which are really just graphs of nodes and arrows. The nodes (which we will call "states") represent a set of behaviors, and the arrows represent transitions from one state to another. The transitions are composed of rules (possibly fuzzy ones) that instruct the control engine when to leave one state, and enter another.

The fuzzy rules tell the robot how to deal with low-level things like what to do as an obstacle approaches. The FSM outlines the overall plan the robot is to follow.

4.3 A Behavior-based, Fuzzy Logic Controller

Traditional logic allows us to deduce things from truth statements. For example, given:

IF too_close_to_wall THEN Speed STOP;
IF ! too_close_to_wall THEN Speed FAST;

we could then determine how fast a robot should go. That is, if `too_close_to_wall` is `TRUE`, then speed will be `STOP`, but if `too_close_to_wall` is `FALSE`, then speed will be `FAST`. Traditional logic can take us quite far, but it is quite rigid.

How do we figure out what `too_close` is? Our robot might jerk to `FAST` when it gets away from things, but then jerk to a `STOP` when it gets close to an object. If you interpret these rules with "crisp" logic, then it really only says what to do at a particular point: it says to stop when you get a certain distance from an object. It really isn't much of a controller at all. How would the robot get going again?

In XRCL, we use logic with a slight twist: instead of requiring absolute truth and falseness, we allow shades of truth. Likewise, if a statement is only partially true, then the rule only has partial consequences. In this manner, if we approach a wall, `too_close_to_wall` becomes slightly true, and the speed of the robot slows down just a tad. This way, these two rules above say quite a bit about what to do in many circumstances. Now, we have quite a controller!

In XRCL, the fuzzy rules are presented in the behavior sections of a controller. These behaviors are small nuggets of control code written to accomplish specific tasks, and are designed to be used in conjunction with other (possibly competing) behaviors. Each behavior is composed of general C++ code, and a set of fuzzy rules. Each fuzzy rule takes the form `IF [fuzzy value] THEN [affecter] [amount]`. [Fuzzy value] is a fuzzy real-valued truth-value in the range 0 to 1.

Behaviors execute simultaneously (being implemented via native threads). At any one time, any number of behaviors may be active in the system, with each trying to control the effectors

4.4 The XRCL Environment

This section describes the XRCL tags. As XML dictates, for each starting tag, there must be an associated ending tag. Ending tags begin with a slash (/).

If you are familiar with C++, behaviors are objects. Here are the main tags that are currently implemented:

Main Tags	
XRCL	The main outside tag
ROBOT	Specify the type of robot

BEHAVIOR	The tag which starts a new behavior
OBJECT	Define an object (i.e., neural network, blob, etc)
MEMBERS	Add member variables to the behavior
ARG	Give an argument to the behavior
UPDATE	The code that runs each timestep
ACTIVATE	The code that runs when this behavior is Activated
DEACTIVATE	The code that runs when this behavior is Deactivated
TIMEOUT	The code that runs when a Timeout occurs
Miscellaneous	
INCLUDE	Use this to add C++ headers
LIB	Use this to add C++ libraries
REQUIRES	Currently unused
CLEANUP	Destructor; runs when behavior is deleted
PREINIT	The code that runs when this behavior is created
POSTINIT	The code that runs right after all behaviors are created, and just before the entire system starts
FUNCTION	Define another function (method) for this behavior's use

The XRCL compiler is case insensitive with respect to the tag names.

Functions that may be called from code areas.

Examples:

SpeedEffect(double dSpeed)	Sets the effect on speed this behavior has on default robot
TurnEffect(double dTurn)	Sets the effect on turn this behavior has on default robot
xcActivate(const char *pBehName)	Activate a behavior by name
Csay(char *fmt, ...)	Generate speech from a format string and args like sprintf

Some examples of Objects:

B21R	B21R Robot class from RWI
Behavior	Main behavior class
Fuzzy	Fuzzy variable class with many overloaded operators
Laser	Class for Mobility-based laser sensors
Network	The main neural network class
Speech	Class for interfacing the Festival speech system

The XRCL Compiler, xc, is responsible for turning the XML XRCL code into a machine loadable binary. Of course, there could be many different types of XRCL compilers --- ones that turn meta-code into C++, Java, Java byte-code, or even a graphical XRCL editor.

```
<controller>
<robot type="B21R">elektro</robot>
<behavior name="Goto">
  <arg default_value="0.0"
        type="double">my_x</arg>
  <arg default_value="0.0"
        type="double">my_y</arg>
  <arg default_value="0.15"
        type="double">my_close</arg>
<update>
  static int done = 0;
  if (! done) {
    xcSay("Here I go!");
    done = 1;
  }
  Fuzzy euclidean(0.0, 1.0);
  // degrees on left:
  Fuzzy left(5, 20);
  // degrees on right:
  Fuzzy right(-5, -20);
  double phi = xcAngle(my_x, my_y);
  double dist = xcDistance(my_x, my_y);
  Fuzzy too_left = right >> phi;
  Fuzzy too_right = left >> phi;
  Fuzzy near_goal = euclidean << dist;

  // The fuzzy rules:

  IF too_left THEN Turn -.30;
  IF too_right THEN Turn .30;
  IF ! (too_left || too_right)
    THEN Speed xcSLOW*1.1;
  IF (near_goal || too_left || too_right)
    THEN Speed xcSTOP;
  TurnEffect(!near_goal);
  SpeedEffect( 1.0 );
  // within my_close cm
  if (dist < my_close) {
    xcPrint("\n[I'm there!]);
    xcSay("I am now at the
    specified location.");
    xcReturn();
  }
</update>
</behavior>
</controller>
```

Figure 2: Example XRCL robot controller. This example shows a behavior-based program designed to allow a robot to go to a specific coordinate.

There are currently three interfaces: an X mode, a curses mode, and a standard console mode.

The X mode allows the most flexibility. For example, you can view sensor readings as rays, boxes, or a circumference (Fig.3.):

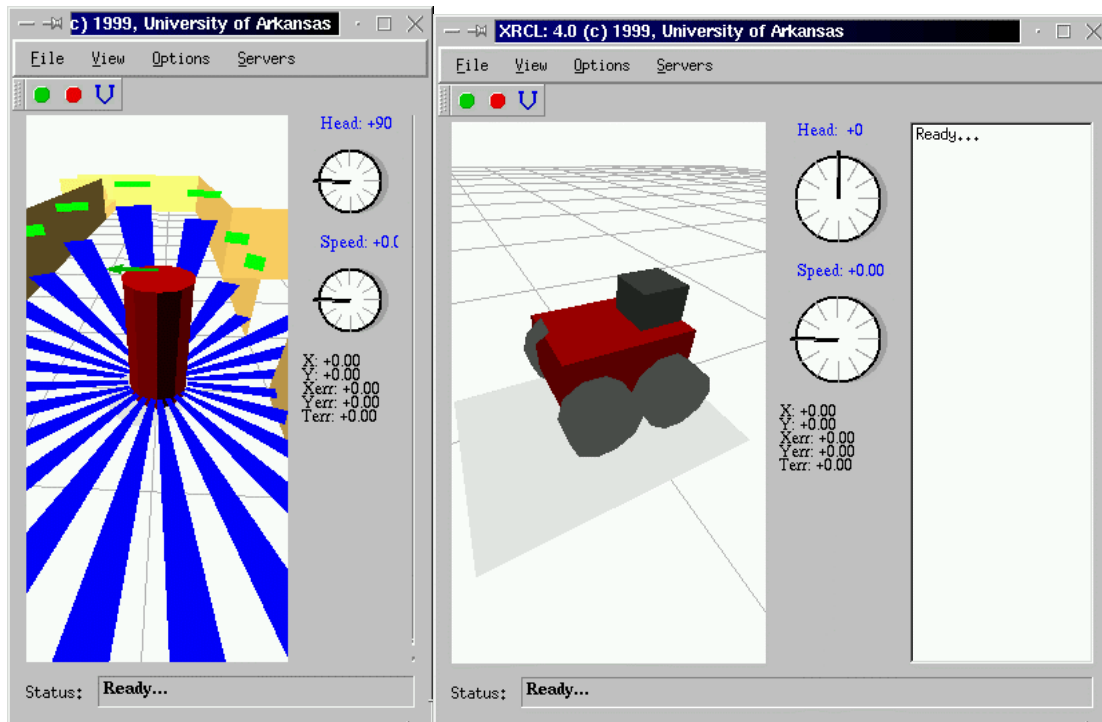


Figure 3. X-mode interfaces

5. POSSIBLE USE OF XML IN WORKPARTNER PROJECT

5.1 Introduction to the project

WorkPartner is a new type of lightweight, futuristic, service robot designed for outdoor use. It can carry different tools and work interactively with humans by learning at the same time the details of the task. The user or operator can be physically present on the same site as the robot and communicate with it by a wireless control unit, or he can use telepresence from a distance place and communicate via Internet. In each case communication takes place via a new generation user interface based on multimedia and cognition.

Mobility is based on a hybrid locomotion system, which combines benefits of both legged and wheeled locomotion to provide at the same time good terrain negotiating capability and large velocity range. Figure 4.a) illustrates the mobile platform, called Hybtor,

on which the manipulation and tooling system is built. The platform has an active body joint and four legs equipped with wheels. The weight is about 200 kg, including all mechanical components and the components of the energy system, the actuating system and the computing system. The payload is about 40 kg, which is mainly taken by the two-hand manipulator system. The purpose of the hybrid locomotion system is to provide good rough terrain mobility and a wide speed range for the machine by using the same mechanics



Figure 4. a) Hybtor platform

b) WorkPartner robot

In order to do work WorkPartner robot will have a two-arm manipulator system. It will consist of a two degree of freedom body, two 5 degree of freedom arms with a gripper and a two-degree of freedom camera head. It will look like a human upper body, shown in the Fig 4.b above. With the manipulator installed into the front of the body, WorkPartner will look like a centaur, so instead of humanoid it can be called centurion robot.

Working tasks require an advanced control system, especially as to force control, therefore the manipulator has a separate control system which is connected to the main control system of the robot. Working tasks are performed with a help of camera information autonomously or interactively with the operator.

With two arms manipulator the robot is to be able to operate with large objects, the sizes of which exceed the arms grippers' size. This is important in many service tasks. The manipulator can handle objects up to 10 kg at 1 meter range. The gripper can be replaced with working tool, like cutting tool.

User interface and the communication with humans will be an interactive concept, which utilizes commonly observed virtual world. The goal is to develop a user interface and control language way beyond the today's interfaces and languages. The MMI will be designed for operators working outside the robot, but in some cases very closely co-operating with it. The robot can also be in a remote place and accessible via Internet.

5.2 Need for Universal language

WorkPartner contains several subsystems like described above; platform, manipulator, operator interface and also some other systems, like navigational system, sensor systems and so. The problem is to put these subsystems work together, controlled by some higher-level program. Also here-developed systems should be easily used and implemented to another projects.

The platform is more than a wheeled machine; it can be controlled in 6 dof inside the area limited by the legs. This means that the control language should be able to take account both wheeled vehicles and legged robots.

The manipulator is human upper torso-like and has all together 9 dof, so the control language for it will be very challenging task to create.

We definitely need internal communication/controlling language between the subsystems and also to program some behaviors to the robot. Also we need a language to which the operator commands are interpreted and sent to the robot. A well-defined language between the subsystems, and researcher and developers, would ease the task a lot when everybody knows the interfaces and the logic's between the subsystems.

The presently developed code is only the basic controllers for the platform. The goal is that someone/something could drive the robot by controlling everything on it. The final goal is just to show the task for the robot, and it will perform it. The short-term goal is to put the subsystems communicate with each other by some simple, easily understood English-like language.

5.3 Control language for the platform

The platform can be driven just like a wheeled vehicle, controlled like a legged machine or it can move using a mode in between these modes, called Rolking (Rolling-Walking). In the Wheel and Rolk modes the curvature is steered while in walking the direction and the yaw of the body is controlled.

Common controls for all these modes are the attitude and height control, and possibility to change controllers on the legs online.

Also a lot of measured data can be transferred from the platform to the other systems.

Here are some parameters that can be controlled:

The mode:	wheel/rolk/walk
Wheel drive:	the body speed/position/force (the wheels), curvature, wheel base, track width, suspension type,
Rolk:	the body speed/position/force, curvature, gait type, (track width)
Walk	the body speed/position/force, direction, yaw, gait type, (track width)

Common:	Attitude ctrl mode (Terrain follow, Auto level, Off) Attitude: (roll, pitch, height), The pos / spd / force controllers, or the 'hardness' of them
Work mode (standing on a spot)	
Leg by leg	
...	

6. XRCL VS. MATH

Programming is much like mathematics, and this follows to the question is there any relation to OpenMath and MathML definitions.

MathML is not programming nor calculating languages, it is being developed to present mathematics formulas and notations on the web. It resembles latex or similar tag based publication languages.

OpenMath could be used here while it is a common notation, which can be used in and between different mathematical programs.

(Another presentation in this seminar)

7. CONCLUSION

Will there be a standardized programming language in robotics, or at least a standard method to present a code. We'll see. The XRCL project is the beginning. Not many (only one?) group is using it and not yet any commercial or industrial vendor is utilizing it but definitely there is a need for such standardization. XML is emerging to be a generalized markup language in all branches of industry, research and in other IT areas, so why not also in robotics.

The WorkPartner project definitely needs a common, well-defined control and communication language. XML provides a good platform to do this. The problem is just to define everything at once so that the work could then continue without interruptions from the communication, or interface, side.

8. REFERENCES

Blank, D.S., Hudson, J.H., Mashburn, B.C., Roberts, E.A. (1999). **The XRCL Project: The University of Arkansas' Entry into the AAAI 1999 Mobile Robot Competition**. Technical Report CSCE-1999-01.

<http://ai.uark.edu/xrcl/>, **XRCL - The Extensible Robot Control Language**, Artificial Intelligence and Robotics, Laboratory University of Arkansas'

Dr. Douglas Blank, Jim Gage, Jared Hudson, Brian Mashburn **XRCL The Extensible Robot Control Language**, Copyright © 2000 by University of Arkansas Artificial Intelligence Lab

Universal Robot Controller™ brochure. Robotic Workspace Technologies, Inc. Florida USA

RobotScript™ brochure. Robotic Workspace Technologies, Inc. Florida USA

ISO standards (not found)

Halme A., Leppänen I. and Salmi S., **Development of WorkPartner-robot – design of actuating and motion control system**, CLAWAR'99, Portsmouth 1999.

Halme, A., Koskinen, K., Aarnio, V-P., Salmi, S., Leppänen, I. & Ylönen, S. **WorkPartner - Future Interactive servicerobot**. The 9th Finnish Artificial Intelligence Conference, Helsinki University of Technology, Espoo, 28-30 August 2000. Helsinki, Finland 2000, Finnish Artificial Intelligence Society, pp. 35-42.

Konolige Kurt, Myers Karen. **The Saphira Architecture for Autonomous Mobile Robots** Artificial Intelligence Center SRI International, 1996