

A Robotic Internet Workstation Design Paradigm*

L. Wyard-Scott[†] R. Frey Q.-H. M. Meng

Abstract

This paper describes a design philosophy developed and currently applied at the University of Alberta ADVANCED ROBOTICS AND TELEOPERATION (ART) Lab. When the philosophy, termed the ‘Robotic Internet Platform’ (RIP) design paradigm, is applied, the result is a teleoperated robot that can be considered an Internet station with the ability to move. Multiple users, with varying roles in system operation, can potentially be connected to the telerobotic system simultaneously. Although the concept is simple, the design implications are great. This paper is intended as a starting-point for developers of telerobotic systems and is meant to aid in several aspects of the design process. It includes considerations that should be made paramount when developing the user interface and the workcell control engine. Two development platforms, TCL/TK and Java, are examined for suitability of use in telerobotic applications.

1 Introduction

Design of a telerobotic system can be a daunting task. Design approaches can quickly become cluttered as more detail is established. A diagram of a typical telerobotic system is shown in Fig. 1. The ease with which the next level of design detail is achieved is based not only upon the task that the resulting platform is to achieve, but also the knowledge that the designer has of technologies relating to the blocks. It is the intention of this paper to expand the designer’s knowledge to include methods that will likely make the design process simpler.

The basis of this knowledge lies in an alternate general view of telerobotic systems. This view is summarized in what has been coined the ‘Robotic Internet

* This work was supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada under grant OGP170446 to M. Meng.

[†] All authors are from the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Alberta, Canada, T6G 2G7. E-mail addresses: wyard@ee.ualberta.ca, rdfrey@ee.ualberta.ca, max.meng@ualberta.ca

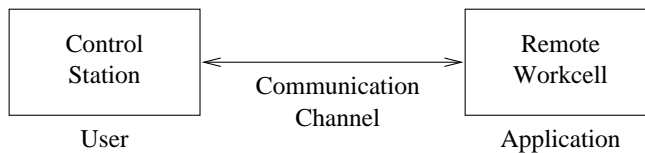


Figure 1: A generic block diagram of a telerobotic system.

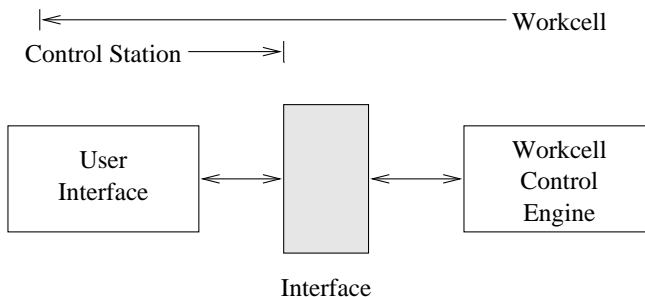


Figure 2: An alternate diagram consistent with the RIP paradigm.

Platform' (RIP) paradigm. Under the RIP paradigm, it is assumed that the communication channel consists of an Internet connection (referring to Fig. 1), and that both the control station and the workcell are Internet machines capable of utilizing this connection. The specific technology and protocol used to achieve this connection is not important at this level. Inherent to the design paradigm is the ability to develop telerobotic systems that have more than one user and/or more than one workcell.

Essentially, the remote workcell can be viewed as an Internet server that has the ability to move, whether in respect to some global coordinate frame, or through movement of manipulators, etc. This concept, although simple, leads to a world of possibilities limited only by imagination. The addition of motion to an Internet server (or conversely, Internet connectivity added to a robot) opens up an area that is just starting to be explored.

The RIP paradigm provides guidelines for implementing this kind of telerobotic system. An alternate block diagram of a telerobotic system that is consistent with the design approach is shown in Fig. 2. A primary goal of this paradigm is to separate the user interface (UI) from the workcell control engine, a separation which manifests itself in the telerobotic system software. The separation of the control station and the workcell is no longer solidly defined; the bounds of where specific features of the system are implemented depends upon:

- the method chosen to implement the UI; and
- the required nature of the workcell control engine.

The workcell control engine is responsible for low-level I/O with sensors and actuators. Depending upon the application being addressed, it may also be responsible for implementation of control algorithms requiring reasonably fast execution speed. It is assumed here that the workcell control engine will be entirely implemented on the physical workcell.

The primary goal of the RIP design philosophy is to develop a standardized workcell control engine that can be utilized by any telerobotic system UI. Conversely, this will allow any UI developer to access the features of the workcell through a standardized interface; something which is essential in a multi-user environment. Additionally, such a separation will minimize development time for researchers and developers.

2 Design Implications

Changing how a telerobotic system is viewed as is suggested under the RIP paradigm changes the design process. A great deal of the design effort needs to be spent on creation of the software interface between the UI and the workcell control engine.

The fact that the Internet is utilized as the communication medium between the control station and the workcell makes this software interface a necessity. In order to make the telerobotic system usable by the widest range of users, no matter their role, it will be necessary to accommodate various UIs written on various development platforms; thus, the need for a strictly defined (yet flexible) software interface arises. The fact that multiple users may be in communication with the robot simultaneously also requires a great deal of planning in order to maintain system integrity.

The interface should receive and issue nothing but ASCII characters. The primary reason for this is that strings can encapsulate other data types (by use of their string representations), and, therefore, offer a sense of global portability. Almost all programming languages deal with strings in an identical way.

In essence, the RIP paradigm emphasizes the need for modularity of the two major system software components – the UI and the workcell control engine. This decoupling is commonplace in software design, and it is suggested that the lessons learned there should be applied to telerobotic systems. When decoupled, telerobotic systems used for research and development may be adopted by institutions with few, if any, changes to the workcell control engine.

3 User Interface Development Considerations

The success of telerobotic systems intended for a wider user-base than simply the developer will depend largely on the user interface. The RIP paradigm assumes a multi-user environment, and a range of user skill levels can be anticipated. The choice of software tools is heavily influenced by these considerations, as well as more traditional concerns such as hardware requirements and ease of development. We approached our selection of UI development tools with the following factors paramount:

- hardware independence/portability;
- ease and speed of development;
- bandwidth/hardware requirements;
- security; and

- overall suitability to telerobotic applications.

Hardware independence describes the ability to quickly port software to new workcell configurations as well as the ability to run the same user-end software on multiple workstation types. Ease and speed of development is especially important in the experimental phase, when the developer needs to make rapid changes to the software and quickly simulate various environmental conditions. Bandwidth and hardware requirements may be an issue when power or circuit real estate is at a premium, such as in small telerobots. Security becomes essential when multiple users, perhaps with various abilities to affect the system state, are able to access the system.

3.1 Tool Command Language/Tool Kit (TCL/TK)

TCL/TK is an interpreted script language that combines elements of a command-line interface and a more friendly graphical user interface. A graphical control scheme can be easily created; however, commands can also be entered interactively and in real time via a command shell. The graphical interface requires the distribution of an application and an interpreter on the control station: interpreters exist for many platforms, but must be installed independently of the application itself.¹ (Alternatively, TCL/TK provides the opportunity for WWW-based delivery: see Section 3.3).

The benefits of TCL/TK are many in a telerobotic application, especially during the design phase. The language provides rapid and simple development of graphical user environments, which can be easily manipulated either automatically or manually. The developer can manually invoke methods contained in code that is running; a boon for environment simulation or testing procedures. TCL/TK also contains many built-in features that are well suited to typical telerobotic tasks. For applications that require low bandwidth, TCL/TK may fit the bill due to its string based character: virtually all communications are short ASCII strings. Finally, unlike shells, the use of TCL/TK allows some separation of the user interface level from the workcell control engine as described in Section 4, due to its heavy reliance on generic strings for inter-process communications.

Some disadvantages are associated with TCL/TK. Generic C routines on the workcell, as defined by an interface specification, can be invoked by a TCL/TK process; however, sockets to the workcell must be manually opened and maintained, somewhat eroding the UI/workcell-control division. Security is not implicit in TCL/TK and would have to be explicitly implemented.

3.2 Java

Java is an hybrid compiled/interpreted language that has object-oriented design procedures implicit in its structure. As such, it is well-suited to complex tasks such as telerobotics that need to be expandable and easily evolved. Machine independence is assured by Java's reliance on the Virtual Machine paradigm pioneered by IBM [Creasy]. Java virtual machines are available for all major platforms, and are included in the kernels of many newly released OS versions.

¹ However, TCL/TK can be linked into an application via its C library.

Furthermore, the user interface offered by the Java platform can be as sophisticated as one wishes it to be: Java is thus well suited for final-version distributable versions of the control software. Other issues stemming from the RIP paradigm, such as security and data consistency, are also easily addressed in Java.

Although platform independence is one of the most publicized virtues of Java, its true strength from a development standpoint is its object-oriented nature. With a well-defined software interface to the basic robotic functions (which can be implemented as C functions and accessed through Java Native Interface calls), the physical robot can be regarded by a software design as just another object. If a Java layer is built on top of the interface, software interaction with the physical machinery of the robot becomes completely transparent to the UI developer. Instead of polling the robot continuously for its status, the developer can regard physical events (such as encountering an obstacle) as software events (which invoke a routine). Communication between the control station and the workcell is also transparent, taking the form of ‘remote method invocations’. The Java VM, in essence, handles all of the low-level communications chores such as opening sockets and verifying information, and the programmer merely invokes methods known to exist within the remote application. This high level of decoupling ensures that evolution of the telerobot hardware has no effect on the control software, so long as the interface integrity is maintained. Additional functions added to a telerobot are dealt with on the control side by adding event-listening modules to the existing software.

Java presents some disadvantages to the designer as well. Hardware requirements are high, and execution speed is low. (There are, however, ‘Java Chips’ on the horizon that implement the Java VM in silicon. These promise to ameliorate performance-related limitations of Java.) Java is extremely general, and has no particular relevance to robotics applications: programming the actual application thus requires more effort. In the same vein, changes to any control program require the program to be recompiled before testing, and inconvenience for real-time testing.

3.3 Web Browser

The concept of using a web-browser as a front end for an application is appealing for two primary reasons. The first is familiarity of the interface: as the WWW gains popularity, the Netscape look-and-feel is becoming second nature to most Internet-connected users. The second is software distribution. By allowing the browser to retrieve software dynamically, distribution becomes moot, and improvements can be made to the software without worrying about legacy applications continuing to be used.

That said, the ‘Web Browser’ interface is not a development platform, but a target channel. Applications developed using either TCL/TK (via a Netscape plug-in) or Java (via native support in most popular browsers) can be distributed via the WWW. If web support is desired by the telerobot experimenter, Java is suggested as the best platform, as no additional support need be added to most browsers, and security issues are more easily handled.

4 Workcell Control Engine

The workcell control engine has several roles:

1. to provide access to the robot sensors and actuators (low-level I/O);
2. to provide feedback control of the actuators (if required by the application);
and
3. to provide the software interface to the UI.

The typical method of providing low-level I/O is through routines written in C or Assembler. The interface to the workcell control engine needs to conform to one major restriction: the data that is initiated by- and destined for the UI needs to be compatible with data types supported by the control engine development platform. Since one of the major motivations of the RIP paradigm is to allow development of a UI on almost any platform, this interface needs to cover as many bases as possible. This requirement points to the use of ASCII characters (or sequences of characters) as the data type of choice.

Certain levels of autonomous operation, survival reflexes, etc. may be implemented within the control engine, depending upon the application being addressed. The issue of how complex (how high-level) the workcell control engine should be is still a topic of discussion (see, for instance, [Lumia (1994)] or [Le Rest et al. (1994)]). The manner in which this is addressed indirectly points to the syntax of the teleoperation language.

The implementation and type of control-loops that are embedded into the control engine may place requirements on where the control engine is implemented. For instance, a servo-loop that requires a fast sampling rate would likely be best implemented directly on the workcell in order to avoid any (potentially non-constant) delay presented by the networking scheme.

The choice of how (and where) the control engine is implemented also depends on the developer's preference. It may be convenient to develop the control engine using either TCL/TK or Java. Accessing low-level C routines in either of these platforms is made relatively straight-forward through use of their well-documented C APIs (or Java Native Interface). This approach may be preferred by the control engine developer because it allows use of features that are not conveniently implemented in C, such as string parsing.

4.1 TCL

Implementation of the workcell control engine utilizing TCL and low-level C routines offers a few unique advantages. First, TCL's primary data type is the string which allows natural adherence to the primary requirement placed on data-types. Second, interfacing with TCL's C API can be more-or-less automated with the use of the 'Software Wrapper Interface Generator' (SWIG) [Beazley (1997)].

Another advantage of using TCL for control engine development arises when TCL/TK is also used to implement the UI. The interpretive nature of TCL, and the ability to call the interpreter to execute commands passed across a socket connection, in a sense makes TCL a teleoperation language in its own right. A disadvantage presented by the use of TCL is its lack of built-in security.

5 Case Studies

VORTEX-PILOT [Le Rest et al. (1994)] is a telerobotic language which exhibits some of the features required under the RIP paradigm. Most of these features, such as using an object-oriented approach to permit expandability, have arisen simply out of a healthy design approach. In light of the RIP paradigm, though, simultaneous connection with multiple (different) users may force a restructuring of their software interface. A similar observation can be made regarding NASREM [Lumia (1994)].

The Unified Telerobotic Project Architecture Project (UTAP) [Leahy et al. (1994)] also emphasizes the need for system design modularity, but not from a multi-user perspective.

6 Conclusions

The true excitement of the RIP paradigm is not the design approach itself but what enters the realm of possibility when it is applied. Although system modularity has been emphasized in many projects, it is believed that the RIP paradigm will shed light on aspects of telerobotic systems that need attention: multi-user, multi-programming environment, and development of a software interface that allows connection to user interfaces written on arbitrary platforms.

Two languages have been examined for use in telerobotic systems: TCL/TK and Java. It is suggested that TCL/TK is well-suited for prototyping and system-level debugging, but that Java and its object-oriented nature are likely more suitable for the end-user software. Thus, if a system is used for nothing but research (where modification to the system software is inevitable) TCL would likely be the best choice. In an established product that, for instance, places system software in firmware, Java would likely provide the most efficient path.

References

- Beazley, David. (1997). "SWIG Users Manual Version 1.1." University of Utah and the Regents of the University of California.
- Creasy, R.J. "The origin of the VM/370 time-sharing system." *IBM Journal of Research and Development*, volume 25(number 5), 483–490.
- Leahy, M.B. Jr. and Petroski, S.B. (1994). "Unified telerobotic architecture project program overview." In *International Conference on Intelligent Robots and Systems*, volume 1, pages 240–244.
- Le Rest, E., Marcé, L. and Rigaud, V. (1994). "VORTEX-PILOT: a top-down approach for AUVs mission telerobotics language." In *IEEE Oceans Conference Record*, volume 2, pages 102–107.
- Lumia, R. (1994). "Using NASREM for telerobot control system development." *Robotica*, volume 12:505–512.
- Welch, Brent. (1995). "Practical Programming in TCL and TK." Prentice Hall PTR.