Adaptable Web Interfaces for Networked Robots

J.-A. Fernández-Madrigal, E. Cruz-Martín, A. Cruz-Martín, J. González, and C. Galindo
System Engineering and Automation Department
University of Málaga, (Spain)
e-mails: {jafma,anacm,jgonzalez,cipriano}@ctima.uma.es; elenacm@isa.uma.es

Abstract — Most research in networked robots that use web interfaces for robot control has been focused recently on the network part, since ethernet involves poor (unpredictable) time performance. However, we believe that the problem to be addressed is more general and should not be restricted only to communication engineering: the interfaced system as a whole should adapt to get the most from the user, from the connection, and from the robot, even when no strict performance is possible. For that purpose, this paper introduces a new architecture for web remote operation of robots that exhibits a high degree of flexibility in its adaptation to each particular user (through modular, configurable JAVA Applets), to the system time-varying performance (through probability-guided, run-time adaptation of control loops), and to the robot software architecture (the standard CORBA is assumed as its middleware). Our approach constitutes an initial step for adapting comprehensively to all the mentioned issues, hence permitting to be employed in very different scenarios: real-time control, telecare, remote surveillance, etc.

Keywords— Human-Robot Interfaces, Networked Robotics, Telerobotics.

I. INTRODUCTION

Networked operation of robots is present in a large number of situations: telecare robotics [1], telesurgery [2], underwater operation [3], space operation [4], etc. However, although interfaces for remote human-robot interaction and control are a technology studied for a long time [5], only recently the world wide web has been considered as the support for the client side, due mainly to its pervasivity and the increase in software and network performance during the last years.

Usually, the main source of difficulties in achieving correct robot control through web interfaces is considered to be the transmission delays, jitter, and lack of guarantee on the bandwidth availability over the ethernet [13]. However, this reasoning has been questioned recently since although no hard real-time is possible, a reasonable degree of dependability can be achieved, for example through specialized communication protocols [6].

These results have inspired us to take a step forward: we believe that the success of a web interfaced robot does not reside only in taking full advantage of transmissions, but in being flexible in a broad sense, that is, the interface must adapt the best to: a) each user, b) each robot operation and application, c) the robot architecture, and d) the time features (that are time-varying) of the whole system. More precisely:

a) User adaptation needs an interface with flexible configuration which enables different skills for operation. In other words: the control loops of the robotic system must be reflected in the interface in different manners for different people. For example, an interface that shows a lot of information can be suitable for precision tasks, but cumbersome for home surveillance by non-expert users.

b) The operations the robot is able to carry out (the application) also define the web interface configuration. For example, a robot in charge of telesurgery tasks needs high precision and smooth, small movements, which imposes different control loops from a robot for caring elderly people.

c) Usually a diversity of robots can be available for implementing a given application, with architectures usually not prepared for web teleoperation. Therefore, an interfacing system capable of integrating with diverse software architectures is desirable.

d) Time-varying behavior of the components of the system (user reaction, data translation algorithms, transmissions, etc.) set obvious constraints in the real-time operation of the interfaced robot. Network transmission is important, but it does not produce the only time delays in the loop.

We have found that up to now the mentioned issues (a)-(d) have not been taken into account within a comprehensive framework [1,6,7]. Therefore, we propose a new architecture for web interfaces aimed to achieve maximum adaptability in all these aspects. Our interfaced system, which is defined as the composition of the client-side web-interface, the network, and the robot, is implemented following these guidelines:

- User’s Environment and Application Adaptation (a)-(b). The client-side web-interface is implemented as a Java applet [8] (the portability of that technology enables maximum adaptation to the user’s computing environment) composed of a set of sensory and actuator elements that link to the system control loops (section II). The control loops linked to the interface are also associated to maximum-loop delay values for enabling further monitoring of control satisfiability.

- Robot Architecture Adaptation (c). The robot architecture is assumed to be implemented upon the
well-established CORBA middleware [16], which has a wide use at present, allowing us to connect to different robot software architectures with a minimum of modifications.

-Run-Time Control Loop Adaptation (d). All time-varying components of the interfaced system (user reaction-time, data translation algorithms, network transmissions, CORBA processing, etc.) are included in a probabilistic model in order to predict the behavior of the system (section III). These predictions are used for the web interface to adapt automatically the activation state of its elements to the time response of the system. Furthermore, different control loops can be automatically selected when the current one is not able to satisfy its time requirements. An analogous work in that direction is [6], however they do not capture uncertainty and only take into account the network transmission time (in addition they assume that the robot-side control architecture is deeply modified to cope with control loop switching, losing some generality).

-Efficiency. Flexibility often leads to inefficiency in execution, thus efficiency is an important point to take carefully in our design. Apart from efficient implementation of algorithms, which is not considered in this paper, direct dedicated TCP connections between the interface and the robot software architecture avoid multiplexing and high-frequency connection/disconnection issues, obtaining far better performances than HTTP or other web protocols. In addition, simple algorithms for control loop adaptation are used (see section III).

These decisions lead to a complete solution that covers most of the flexibility issues commented before. The works in literature that we have found similar to ours (for example [6],[7]) either do not take into account the influence of the whole system in its run-time performance (typically, only the network communication is considered), or are not general enough for coping with different robot architectures (they typically couple the interface to a particular robot). In addition, there is no systematic way of linking to the interface the control loops of the system along with their time requirements.

In the rest of the paper, our solution is described in more detail. Section II focuses on the client-side web interface configuration, section III takes the run-time adaptation of control loops in consideration, and in section IV we show an implementation example in a real system. Finally, some conclusions and future work are outlined.

II. WEB INTERFACE CONFIGURATION

Each user has different necessities and skills, so different web interfaces should be used for different people (even for the same application). In addition, the interfaced system, including the web front-end and the controlled robot, is assumed to provide a number of operations, that depend on the particular robot and scenario, which also determine the interface configuration. For example, the same mobile robot could be used for both elderly care or house surveillance, and different web interfaces should be constructed for each case.

In order both to enable maximum configurability of the web interface and to achieve optimal adaptation to time-varying behavior of the system, we construct the client front-end upon a set of modular widgets\(^1\). Each widget constitutes a link to some signal of one of the control loops existing in the system, providing access either to actuation or to sensory data (see fig. 1). For example, for direct control of a robot arm, a sensory widget that displays the sensor readings (of a camera, for instance) and an actuation widget for changing the position of end-effector may suffice, although more widgets could be used for the same purpose. We also allow the use of passive widgets in order to facilitate user interaction (for instance, a passive widget to display a static, pre-built map of the environment for guiding navigation of a mobile robot).

This modular framework allows us to select the group of sensory/actuation/passive widgets of a given control loop that are more suitable for a given application and user. The detailed scheme of control formed by this selection is shown in fig. 2: we consider that user’s actuation (on a given actuation widget) generates some service requests that are transmitted through the network to the suitable modules of the software architecture of the robot; once the requests are completed, their return data (if any), plus the readings from the sensors associated to the sensory widgets of the control loop (obtained also by requesting some services from the robot), are returned to the display. Although the information of service requests associated to each actuation widget is currently hard-coded into the interface, it should not be difficult to develop in the future a system that automates the generation of such code.

Notice that in some situations, fitting every element of the interfaced system into the scheme of fig. 2 may be too restrictive. For example, we might need to refresh some sensory widget periodically instead of after user actuations. For that purpose, we permit to add a refreshing period to any sensory widget, which will not be considered as part of any control loop.

When the desired widgets are configured, additional information is associated with the represented control loop. Mainly, a maximum delay time \(T_d\) is given for each loop that indicates the maximum time desired for closing the loop: from the “receive user’s action” to the “display

\(^1\) Typical components in a graphical interface are usually called widgets: buttons, panels, etc.
process” in fig. 2. That value along with some others ($P_C$, $T_F$, and $T_W$ explained further on) are used at run-time for tuning of data flow and control loop activation/deactivation, as described in section III.

III. RUN-TIME ADAPTATION OF CONTROL LOOPS

Our architecture automatically adjusts, during robot operation, the web interface configuration for reducing the time consumption of the system, in order to provide the human user with the most reliable and comfortable control. This adaptation is achieved at three levels of abstraction, that are (top-down):

1) Coarse Adaptation. The system can activate at any moment the most appropriate control loop for the robot (deactivating the current activated one), among the ones configured in the interface. This allows the user to control at different time scales, ranging from a tight real-time control (lowest level) to a loose task-based control (highest level). For that purpose, the system is permanently monitoring the time spent in all the parts of the loop: user reaction, network communications, software processing, etc., for feeding a probabilistic model that allows it to decide whether the time requirements for the current loop ($T_D$) are likely to be achievable or not in the next future.

Also, the user can activate any of the available control loops at any moment simply by acting on one of its actuator widgets (the current loop is automatically deactivated).

2) Medium Adaptation. With the time probabilistic information mentioned before, the web interface can also enable or disable individual sensory widgets that are shown to the user, without deactivating the current loop. Consider, for example, that two or more sensors are available for monitoring: maybe the sensory widget corresponding to the most time-consuming one can be inhibited before switching to a different control loop.

3) Fine Adaptation. Also before giving up with a given control loop, its time requirements can be relaxed by adjusting the amount of sensor information that can be obtained from the robot, without disabling any widget of the interface. For example, one of our 360-degrees radial laser scanners can provide up to 4096 points (about 32Kbytes transfers through the network), but not all these data must be transmitted to the web interface in order to obtain a good control. Similarly, the image transmitted by a video camera can be compressed or reduced. These actions relax the time requirement of the loop, improving control.

By using these three kinds of actions the interfaced system is able to adapt its behavior to work correctly under very different –and variable- conditions (slow network connections, different users with different skills, etc.). In the following subsections, the probabilistic time models we have identified for the different parts of the interfaced system and the details on implementing our probability-guided, run-time adaptation of control loops are explained.

A. Control Loop Time Modeling

In the following we describe in more detail each part inside the loop shown in fig. 2 (that represents any control loop configured in the interface), and identify the probabilistic models for their time consumption:

-User Reaction. Modeling the time consumption of user reaction is a difficult issue. Usually, human reaction-time is approached by extracting a single value from a set of probabilistic measurements [11], which is clearly insufficient in most applications since human reaction-time depends on several factors, as varied as: amount of information interpretable by the user, spatial arrangement of that information [12], rate of change in sensory data, etc. In general, it is acceptable to take the human reaction-time as a gaussian distribution.

-Translation. Both graphical sensory data that is displayed and user actuation on the graphical interface must be translated from and into formatted messages that flow through the network. Also, messages received at the robot side must be translated from and into requests from services of modules of the robot software architecture. The time consumption of these operations depends basically on: the size of the data, the computational complexity of the translation algorithms, and the CPU scheduling provided by the operating system. The first two sources of time complexity can be approximated well by linear functions. In non-real-time environments, sporadic high time consumptions can appear due to the third source. We have modelled these anomalies by adding exponential distributions when necessary.

-Network Transmission. This part of the loop represents both the physical network, queuing buffers, and the OSI protocol processes. Assuming the use of TCP connections implemented over an ethernet network, the time delays for transmission are not bounded. In literature it has been stated that the arrival time of ethernet communications tends to
a Poisson process as long as the network is fast enough, thus the interarrival time can be modelled as an exponential distribution [13]. In cases where the network is slower the Poisson hypothesis is no longer valid. Experimentally, it has been shown that then it is more appropriate a beta distribution [14]. Therefore, the network transmission time consumption can be modelled as exponential or beta, since they are the dominant components of that delay. Our experiments in different environments (twisted-pair networks, modem, wireless) agree with all the mentioned results (see fig. 3).

-CORBA Processing. Requests of services of the robot’s software modules (that we assume implemented as CORBA objects) involve a sequence of procedures in order to move the requests (and answers) to the appropriate processing code. This is performed automatically by the ORB - the inter-object communication core of CORBA- and the POA -the intra-object management component of CORBA- by marshalling, demarshalling, queuing, threading, etc. [9]. The time consumption of these operations depends basically on: the size of the input/output data of the method being requested, the algorithms implemented in the ORB and POA, and the real-time capabilities of the operating system. Modeling such situation is analogous as in the above paragraph about translation.

-Service Processing. When the input data arrives to the corresponding service of a module in the robot side, it is processed by some robotic algorithm, and possibly some output data is returned. Due to real-time restrictions on robotic applications, robotic algorithms are typically of polynomial time complexity. Thus, we can assume that the time consumption of the service depends polynomially on the size of the input/output data. Again, if the operating system is not real-time, an exponential distribution is added for time anomalies.

-Display Processing. The sensory data returned by the robot to the user interface must be displayed appropriately. Most of the algorithms for displaying data are of linear complexity on the size of the data, or at most polynomial. Time consumption anomalies can then come from the non-real-time characteristic of the client-side computer, which is the most common situation in any user environment.

For obtaining the final time model, polynomial and linear processes have been modelled by uniform distributions in order to cope with slight variations in time consumption due to conditionals or in the measurement of time. Due to the Central Limit Theorem [17], and assuming a large number of samples, the sequence of the distributions described above yields a gaussian distribution as the total time consumption of the control loop.

In our measurements, we have also noticed that the user reaction time is commonly the largest time in the loop, and that client-side processing times are also important, which confirms the main motivation of the paper: the network response time is not the only factor to take into account in a web-based interfaced system.

B. Probability-Guided Adaptation of Control Loops

The mechanism for the adaptation is supported by monitoring continuously the round trip time (RTT) [15] of the network and the time consumed in the other parts of the system. The information is recovered by the client-side interface. From the history of these measurements and the knowledge that they should fit a gaussian distribution (as explained in the previous subsection), the JAVA applet

![Fig. 3. Real measurements of internet transmissions (round-trip time/2, in milliseconds) along with their MLE-adjusted probabilistic (exponential and beta) models. Plots are for three different networks: top – ethernet twisted-pair 10/100; middle – wireless 802.11g 100Mbps; bottom – modem 56K.](image)
infers the parameters of their best distribution through the maximum likelihood estimator (MLE) [17]:

\[
\mu = \frac{1}{m} \sum_{i=1}^{m} t_i \quad \sigma^2 = \frac{1}{m} \sum_{i=1}^{m} (t_i - \mu)^2
\]

where \( m \) is the number of time measurements available \( (t_i) \). Other approaches could be implemented, such as Bayesian estimation [17], that are more robust with respect to the initial variability of data. However, computing the Bayesian posterior distribution over the set of all possible Gaussians involves an important computational cost (the 2D-space of parameters \((\mu, \sigma)\) must be sampled and all the points must calculate their posterior each time a new measurement is available). Furthermore, our current approach is not much sensitive to the typical problems of MLE due to the large amount of measurements that its simpler calculation permits.

Having a suitable gaussian distribution for the time delay of the current control loop\(^2\) allows the system to predict the probability of closing the loop in the prescriptive time \( T_D \) (or in a shorter time; thus, we use the cumulative distribution of the gaussian\(^3\)). When it is estimated that in the next future that probability falls under certain confidence threshold \( P_C \), the actions described at the beginning of section III are taken: first, fine adaptation is tried in order to reduce the flow of data without affecting the activation of widgets; if that has no effect within a given safety time \( T_S \), medium adaptation is carried out whenever several sensory widgets are taking part in the affected control loop; if that is neither possible or has no effect within another given time \( T_M \), coarse adaptation takes priority and deactivates the control loop. This can lead to stop the robot if no other loop is available, by launching a predefined critical request for those cases.

![Fig. 4. Evolution over time of the predicted MLE probability that a given control loop can be closed within 1.036 seconds. The oscillations at the beginning (when the estimated gaussian is not well known) rapidly reach a stationary probability of about 72%.

The loops that are not being used can be set by the user to a safe setting point in order to activate their goal in the very moment they take control over the current loop. For example, there can be two loops for controlling robot navigation: one through direct speed/direction commands and another through specification of a destination location in a map (which automatically avoids obstacles and constructs its own path to the goal). The user can start controlling the robot with the former, but for the case that that loop cannot achieve its time requirements, he/she should set a safeguard destination in the second loop that will be taken immediately the former loop is deactivated.

Notice that relying on probabilistic measures in general could produce high-frequency oscillations in the activation/deactivation mechanisms described before, which could be confusing for the user. However, the use of a confidence threshold tends to avoid those oscillations, since the cumulative distribution function corresponding to the estimated gaussian vary at a slow rate (see fig. 4).

IV. IMPLEMENTATION EXAMPLE

In this section we describe an example of implementing our architecture for a real mobile robot. We have constructed an interface system for our mobile robot SENA, shown in fig. 5. The robot is equipped with several sensors (a laser scanner, sonars, infrared, and a camera) and carries an onboard computer with wireless internet connection, which runs the software architecture of the robot. That architecture is composed of a set of modules (CORBA objects) that provide a number of services (methods of those objects). That software has been developed in the last years with a robotic software development system called BABEL [10].

![Fig. 5. On the left, our electric wheelchair SENA. This robot is equipped with laser, sonar and infrared sensors, a camera mounted on a pan-tilt unit, an onboard computer, and a wireless internet link. On the right, the web interface configured for our experiments.

For our purposes, some modules of the architecture have been modified by adding the tasks in charge of connecting to the interface through TCP. The part of the modules that have been modified has been small due to the modularity of CORBA middleware.

The interface is intended to enable remote commanding of the wheelchair for navigation. Two control loops are
included: one for actuating directly on the motors (speed/direction) and another one for navigating reactively (through a potential field method [18]) to a given geometric point in the 2D floor map of the environment. The interface also includes sensory widgets for showing the current speed/direction of the wheelchair, its position, and the readings from the frontal laser scanner.

Fig. 6 shows the path followed by the wheelchair in an experiment in which the user begins by controlling it directly through speed/direction (before that, the user has established the desired goal point in the second loop). At time 17 sec., the interface estimates that the current control loop cannot achieve its time requirements, mainly due to slower user response and network congestion, and deactivate it (coarse tuning), activating the second loop for reactive navigation. The wheelchair continues smoothly to its goal since it is the same as intended by the user until that. Later on (time 26.37 secs.), the first control loop can be reactivated and manual control is resumed until the wheelchair reaches the destination.

Fig. 6. Path followed by our mobile robot under two types of remote control. Empty circles constitute the part of the path commanded by reactive control (loose), while black points are for the direct speed/direction control (tight). The map of the environment is the result of fusing together several laser readings.

V. CONCLUSIONS AND FUTURE WORK

In this paper we have established that remote control of robots via world wide web interfaces is not limited only by the unpredictable time characteristics of the network, but also by other components (the user, the processing of data, etc.). Therefore, if we desire to construct optimal web interfaces for robot teleoperation, not only must we exploit the most of the network performance (which is the usual approach in literature), but also of the other parts of the control loop.

We have proposed here a comprehensive approach in which all the relevant parts of the loop are adaptable in one way or another in order to obtain the optimal performance of the whole system: the user interface can be configured for each user and robot operations, and the real-time performance of the system is monitored in order to produce further tuning that keeps the most of the time the desired control loop active. When that is not possible, other loops can be activated that require less stringent time constants.

In the future, we plan to complete this approach by the addition of multi-user and security issues and obtaining finer time adjustments of control loops by using robot behaviors that admits parameters to reduce their time consumption, for example navigating slower, etc.

REFERENCES


