In this paper we summarize some of the research efforts carried out at the Robotics Laboratory ALTAIR of the University of Verona (Italy) towards the development of a high performance architecture for the teleoperation of remote systems, including force reflection teleoperation, called Penelope (Πηνελοπεια), characterized by an innovative mix of software and hardware features [1]. Penelope is a modular and distributed software architecture, aimed specifically at teleoperation systems. With this architecture, it is possible to control, in a distributed framework, several types of robots in a simple way and test different teleoperation configurations, including new devices, sensors and boards. The architecture supports a mixture of software and hardware components, including I/O boards, FPGA boards and Multi-Axes Controller boards, which can be easily interchanged with a software implementation for testing. Penelope is programmed in C++, using Object Oriented Programming (OOP). In this way, every real object in the framework is represented by a specific object in the software. With this programming choice, it is also possible to use hierarchy, polymorphism, function overloading and all the other OOP mechanisms to develop a complex and modular software system. Furthermore, a structured communication infrastructure is necessary, since a general teleoperation consists in general of a network of PCs controlling different physical devices (master, slave, sensors, GUI, etc.) and often using different Operating System.

The paper describes some of details of the Penelope systems, some of its performance tests, and its use with dedicated hardware to freeze stable software modules, increase their execution speed, and prevent failures due to unpredicted event occurrence. After a brief description of the state of the art in software architectures for robotics in the next Section, the paper describes the software organization of Penelope and its main characteristics. Next, we describe the trade-off of implementing software modules in hardware and the difficulties of having one-to-one replacements of software modules. Finally we conclude the paper with a few considerations on future development of our teleoperation system with specific emphasis on its use in critical applications.

INTRODUCTION

Space teleoperation could give a significant boost to science education and, at the same time, reduce part of the costs associated to space operation, by moving some operation to Earth based stations. However, the entry into this field requires a long and steep learning curve that prevents most of the possible players to take an active role. In fact, the standards are very complex, require dedicated computers and connections, and communication protocols are different from one experiment to the next. Therefore, some of the requirements of teleoperation software must include the possibility to easily interconnect different devices, test different control algorithms, process data from several and heterogeneous sensors and use various connection media. From these needs, the idea of a middleware integration framework arises. The framework has to be designed to enhance the ability to modularize, extend and reuse the software infrastructure in order to handle a distributed environment. In addition, some real-time capability is needed to maintain transparency and stability while data are sent over the communication channel, dealing with unpredictable time delays and low bandwidth. Hidden advantages of such a distributed real-time middleware for teleoperation are reduced systems cost, remote expertise on demand, dynamic access to the system, and increased reliability.

The common approach followed for space applications is to develop a dedicated system to ensure master/slave (or other device) communication. However this solution, which is the natural choice in this critical application, is economically unacceptable in most cases. In this paper we report on the efforts carried out at the Robotics Laboratory ALTAIR of the University of Verona (Italy) towards the development of a high performance architecture for the teleoperation of remote systems, including force reflection teleoperation, called Penelope (Πηνελοπεια), characterized by an innovative mix of software and hardware features [1]. Penelope is a modular and distributed software architecture, aimed specifically at teleoperation systems. With this architecture, it is possible to control, in a distributed framework, several types of robots in a simple way and test different teleoperation configurations, including new devices, sensors and boards. The architecture supports a mixture of software and hardware components, including I/O boards, FPGA boards and Multi-Axes Controller boards, which can be easily interchanged with a software implementation for testing. Penelope is programmed in C++, using Object Oriented Programming (OOP). In this way, every real object in the framework is represented by a specific object in the software. With this programming choice, it is also possible to use hierarchy, polymorphism, function overloading and all the other OOP mechanisms to develop a complex and modular software system. Furthermore, a structured communication infrastructure is necessary, since a general teleoperation consists in general of a network of PCs controlling different physical devices (master, slave, sensors, GUI, etc.) and often using different Operating System.

The paper describes some of details of the Penelope systems, some of its performance tests, and its use with dedicated hardware to freeze stable software modules, increase their execution speed, and prevent failures due to unpredicted event occurrence. After a brief description of the state of the art in software architectures for robotics in the next Section, the paper describes the software organization of Penelope and its main characteristics. Next, we describe the trade-off of implementing software modules in hardware and the difficulties of having one-to-one replacements of software modules. Finally we conclude the paper with a few considerations on future development of our teleoperation system with specific emphasis on its use in critical applications.
PRIOR WORK ON SOFTWARE ARCHITECTURES

Software engineering together with the growth of object oriented languages have given a strong impulse to the development of software architectures capable of managing complex and heterogeneous tasks, i.e. robotic systems. Orocos [2] and Smartsoft [3] are frameworks where execution and data flows, scheduling and data consistency are strictly defined. GenoM developed at LAAS/CNRS [4] allows automatic building, from high level specifications, of modules which are interacting. CLARAty [5], [6] has a two level structure, abstract and hardware related. This structure permits developer to concentrate on a specific aspect without worrying about the interconnections among various elements. New distributed architecture are tailored for mobile and cooperative mobile platforms, as MARIE [7] based on a mediator pattern and MIRO [8] which is CORBA based. Almost all these frameworks are designed for mobile robots, except Smartsoft, consequently with little focus on real-time problems such as control, security and recovery. In MCA [9] is possible to define priority and timing of the various thread by using Linux with real-time extension. In order to have a general-purpose architecture and language, vendor, and operating system independence, CORBA is the solution adopted in the research described here [10]. Deterministic performance is achieved using standard Linux drivers, applications and functions, in the RTAI (Real Time Application Interface) [11] extension of the standard Linux kernel, making it fully pre-emptable. Another enhancement to achieve of deterministic performance is by moving the bottlenecks present in the control loops of the devices to hardware. This can be done by using specific interfaces to fulfill the middleware framework requirements and preserving aspects such as transparency and software independency. Using CORBA, others important aspects such as security and fault tolerance are ensured by using distributed and redundant resource management. Another goal of the middleware development should be to make the software modification process easier.

The research carried out at the Altair robotics laboratory has produced Penelope [12], a modular and distributed software architecture, aimed specifically at teleoperation systems. With this architecture, it is possible to control, in a distributed framework, several types of robots in a simple way and test different teleoperation configurations, including new devices, sensors and boards.

We have chosen to implement Penelope using Object Oriented Programming (OOP), do that every real object in the teleoperation framework is represented by a specific object in the software. A possible approach is to develop a dedicated system to ensure master/slave (or other device) communication [13]. However, to obtain a general and more powerful communication infrastructure, we have chosen to use CORBA. With this communication framework, it is possible to distribute Penelope over different PC stations simply by using the object interconnection service of CORBA (OOP helps in using CORBA in the code). Communication is implemented using the real time implementation of CORBA, The ACE ORB middleware (TAO) [14], which allows to control the communication timing more precisely, and to use different communication protocols or scheduling priorities, when necessary. CORBA provides other important features, such as real-time event dispatching and scheduling, periodic event processing, efficient event filtering and correlation mechanisms, and multicast protocols required by real-time applications in order to provide efficient, predictable, and scalable quality of service (QoS) end-to-end.

All the infrastructure is implemented using Linux under a GPL (General Public License). To respect strict timing constraints, for example in control algorithms, we use the RTAI Linux extension. Another important feature of Penelope is its Graphical Interface independence. Penelope can be used by different types of Graphical User Interfaces (GUI), by means of different user interface builders and different graphical libraries. GUI can run and communicate with Penelope on various OS and also using a web interface (with a new CORBA IDL interface). Each user can implement his own personal interface and use Penelope only knowing the CORBA IDL class interfaces. In the following Section, we summarize some of the main features of Penelope and describe some of the performance tests carried out so far.

PELELOPE FEATURES

Penelope is a modular and distributed software architecture, aimed specifically at teleoperation systems. Because of the complexity of the desired goal, we have designed the architecture of Penelope using the UML Class Diagram shown in Figure 1. New classes can be added easily to extend the architecture to other systems. Penelope main classes are Robot, Board, Sensor, Controller, and Trajectory that are briefly described in the following (the suffix Impl indicates an implemented class).

**BoardImpl**. Represents all devices used as interface from a real robot to a PC station, e.g. to read encoders or write voltage/torque to the motors.

**RobotImpl**. Manages all types of robots. A manipulator, a joystick, a mobile robot or a multiple robot (for example a mobile robot platform connected physically to a manipulator) are sons of RobotImpl class via inheritance.
Fig. 1. Penelope UML Class Diagram. It represents the diagram at the specification perspective and does not consider the interface classes for communication.

**ControllerImpl.** This class contains all the feature necessary to control a robot; for example controller algorithms and controller parameters.

**SensorImpl.** Interface class to different sensor devices. With inheritance it contains the force sensor class, the sonar and laser device class, and the camera device as a visual sensor.

**TrajectoryImpl.** It produces all references followed by a robot, for this reason this class can be used in two different way, as a trajectory generator and as a communication channel between master and slave. Clearly, there are complex relations between these classes, as shown in Figure 1. For example an instance of ControllerImpl (a ControllerImpl object) can be in relation with some instance of other objects. The control algorithm manages a robot and there must be a relation between this two types of objects. A control algorithm can also use a sensor, for example an instance of the force sensor in the case of a force control algorithm.

**Object Communication**

CORBA TAO middleware is used as communication infrastructure. Any main class in the architecture inherits from the corresponding class (interface) in IDL. Therefore, for example, RobotImpl inherits from Robot IDL interface. All interfaces of methods in IDL are implemented in the corresponding real C++ class and can be invoked in a distributed way. CORBA defines the Naming Service to invoke the distributed methods of a specific object and therefore, the architecture knows their references. Penelope uses this service to read and write references from a complex tree managed by the Naming Service.

By means of this service, it is possible to receive any object reference and use it as a distributed object. However, a teleoperation system is more complex and Penelope must be able to create or destroy an object on demand. When the architecture starts, no object is created and there are no references in the Naming Service. Only after the operator sets the objects, the references are created. To allow this feature, Penelope uses the class creatorImpl. When Penelope starts, only this object is active and its reference is in the Naming Service. Methods of creatorImpl can be invoked, to create or destroy any other architecture object.

**Robot Control**

Some features of a teleoperation system have real-time properties. The more interesting example of this is well represented by the Robot class. A robot must be controlled and typically a control algorithm is executed on a Board and not on a PC station, because of time constraints. However, in a flexible teleoperation system such as Penelope, it must be possible to use different, and often new control algorithms. For this reason, board-based control is not the best choice. In Penelope control algorithms run on a Linux PC equipped with RTAI (Real Time Application Interface), an extension of the standard Linux kernel that make it fully pre-emptable.
When the controller is started, two different threads are executed: `communicationThread()`, an RTAI thread that reads the reference input, and `controlThread()`, an RTAI thread that reads the reference in the controller class attribute, computes the correct signal (for example torque/voltage signal) and sends it to the robot using the specific board object.

**Penelope in a Teleoperation System**

Penelope is used to manage a teleoperation system, which typically include multiple robots, sensors or general devices, connected to several PC stations, each one running a copy of Penelope. In any instance of Penelope, only a subset of the system objects is created and the communication is managed by CORBA through the Naming Service and the “creator” object. In the GUI, it is necessary to choose which object is created in a specific Penelope instance, and which distributed object is used by the other Penelope instances. To use them, it is necessary to obtain only the reference through object names stored in the Naming Service. In a teleoperation system, Penelope is robot independent since it can control either Master or Slave devices, as shown in Figure 2.

**Test Results**

In order to test Penelope and evaluate the performance of our architecture, a teleoperation test is presented. To define a demanding test task, we use the maximum number of classes implemented in Penelope in a time critical operation. We consider a simple teleoperation task with force feedback. We use a NASA-JPL Force Reflecting Hand Controller (FRHC) as the system master. The FRHC is a six degree of freedom (dof) joystick with force feedback in every joint. It permits to operate with full dexterity in a cubic workspace of 30x30x30 cm developing forces up to 10 N and torques up to 0.5 Nm. The joystick is driven by a custom designed controller implemented mostly on a FPGA board (NI PCI-7831R). The idea behind this approach is to improve the computational speed of the master while maintaining accuracy and integrating the controller in the architecture. The FPGA is a good instrument to test hardware task implementations at a reasonable cost, reducing the time between the system design and final implementation. Our approach relies on porting the most time-consuming teleoperation algorithms to hardware.

Our FPGA permits to implement in hardware the forward kinematic algorithm, as described in next Section. The slave robot is a Unimation PUMA 200 arm with six dof. We use a virtual sensor to simulate different force/torque profiles reflected to the user. Every movement the operator imposes to the joystick is transferred to the robotic arm with the appropriate workspace transformations and scaling. The readings of the robot sensors are fed back to the joystick and, through the activation of the FRHC motors, to the operator. The communication is carried out on a standard 100MB ethernet LAN, as shown in Figure 2. We run the following tests to “tune” and evaluate our architecture.

1. The first test is to evaluate whether the use of CORBA adds a significant communication overhead with respect to more traditional communication strategies such as sockets. A CORBA version of PING was implemented on top of our architecture and results are compared with the traditional ICMP PING of the Linux operating system. The comparison however, is not completely straightforward. In fact, CORBA supports only TCP and UDP protocols, and we added a similar functionality on top of Penelope using TCP. The test is repeated in our laboratory LAN under different load conditions: with a dedicated point to point link between two computer, with normal intranet
and Internet traffic and with an overloaded LAN, as shown in Figure 5. We can see that CORBA-based PING is slower than normal PING and this difference is easy to see specially with the PC called Hector. This can be explained by considering that the overhead introduced by CORBA has a larger impact on a slower PC than on a newer and faster hardware.

2. The second performance test aimed at evaluating the maximum frequency of the control loop achievable under RTAI Linux, with our hardware set up under heavy system load. We measure RTAI performances with Hector PC with normal load and with heavy system load and with a control period of 2ms. No performance loss was detected in both situations.

3. The last tests were designed to verify the modularity and reusability of Penelope. Different configurations were set up to see how many reconfigurations and subsequent recompiations were needed by Penelope. We first tested a standard teleoperation task in a point to point connection, one PC controlling the master station, and another PC controlling the robot arm, simulating the force/torque sensor and running the CORBA naming service. Then, four computers are involved in the same task: one for every device (robot, joystick and sensor) and one for the CORBA naming service. No recompilation was needed to pass from one configuration to the other. The performance of both systems was comparable under the load condition tested.

Fig. 3. Test communication performance with different PC (Vincent: Pentium 3, 866Mhz, with 256M RAM, Hector: Pentium 4, 2.4Ghz, with 512M RAM). Figures in the first line are with normal laboratory traffic; figures on the second line are under overloaded traffic and on the third line with point to point dedicated connection (Crossed). Black lines represent traditional ICMP PING in Linux while red lines represent CORBA-based PING. All data are expressed in milliseconds.

The tests carried out show that the performance is not degraded significantly by the addition of a complex middleware such as CORBA, and that significant benefits are achieved by the simplicity in the reconfiguration and the flexibility of the overall system, and by the good clock stability of RTAI-Linux operating system.

HARDWARE MIGRATION OF STABLE SOFTWARE

In critical teleoperation applications concern on security, fault tolerance and error recovery are as important as performance and hardware reconfigurability. Therefore, all efforts have to be carried out to reduce the possibility of unexpected errors due to the program logic. However, this is not the only cause of faults. Hardware failure, software timing, equipment replacement, are all possible causes of malfunctioning. Penelope addresses this issue by supporting the migration of software into hardware components, FPGA chips embedded into the single devices, which then are free from many failure causes. To address malfunctions due to communication failure, server congestion, and PC fault and so on, we take advantage of the run-time fault detection and tolerance of the middleware (TAO) used in the communication layer of our architecture, exploiting distributed events handling and service recovery. Furthermore, to
take into account failures due to external events, a carefully designed supervisor layer can be added to the architecture, to handle all the planned situations and react in a safe way to anomalous behaviours.

FPGA are rapidly growing in many application areas because of their synthesis tools and excellent speed. The advantage of a reconfigurable board for design testing takes an important role in lowering developing cost and time. The use of this board even for a final product is justified when the speed of hardware has to be comparable with the need of flexibility and reusability. Signal and image processing and automatic solver represent good examples of FPGA utilization in different fields. In robotics two main approaches arise, the first regarding navigation of mobile robots, the second applied to the control of devices. In the first scenario iterative and parallel algorithms based on neural network, simulated annealing and genetic algorithms are often used to take decision about trajectory generation and obstacle avoidance in real time [15], [16]. In [17] an FPGA-based CAM-Brain Machine (CBM) is used to control behaviors of a life sized robot kitten. In the control theory application closed-loop algorithms have been studied and implemented in an FPGA. In [18] a parallel PID design is used to completely control a wheelchair; in [19] different PID implementations in reconfigurable hardware are evaluated and used. A parallel PID algorithm with fuzzy gain controller is proposed and tested in [20] while in [21] a small robot is driven by an FPGA-based PID controller that makes the system completely autonomous despite the small size of the robot: different strategies and possible implementations are presented.

In our application we use a FPGA board (NI PCI-7831R) to control the teleoperation master, a NASA-JPL Force Reflecting Hand Controller (FRHC). An Unimation Puma 200 equipped with f/t sensor is used as the system slave. The FRHC is a six degree of freedom (dof) joystick with force feedback in every joint. The goal of this approach is to improve the computational speed of the master while maintaining accuracy and integrating the controller in the architecture. Our approach relies on porting the most time-consuming teleoperation algorithms to hardware, such as the forward kinematic algorithm and, in the future, the force feedback calculations, together with some local model for realistic haptic feedback. The board handles both the low level tasks and the algorithmic part of the problem in a transparent way with respect of Penelope. On the FPGA we have the low level part that is the logical domain of the abstract class board, whereas the algorithms are in the robot class. We decoupled the access to the same board using two different classes. In this way we avoid rewriting the code and let the invocation of methods through CORBA follow the logic used by normal couple robot/board. The hardware used is a Nation Instruments [22] FPGA based on Xilinx Virtex II chipset. The board has 1 million gates, 5120 logic slices, 81920 B of memory, 96 digital I/O ports and 8 analog I/O ports. The only way to access, program and synthesis code on the board is to use National Instruments LabVIEW FPGA [24] available on Windows platform only. LabVIEW is an easy to use software that permits the utilization of an FPGA without knowing VHDL programming language and providing high level program languages statements. The drawback is that it limits the possibilities of code optimization and the use of low level commands. No communication between the board and the host PC is available outside LabVIEW.

To quantify the performance of the hardware migration of the software algorithms, we consider three parameters: resource utilization, speed and power consumption. We use the programmable board on an ATX PC so power consumption is not an issue. The others two parameters are checked using the report and the timing tools that National Instruments LabVIEW FPGA provides.

1. Resource utilization: One important issue about FPGA utilization is the available area in which the synthesized code is stored. For example, for the forward kinematic algorithm the resources used consist of 27 multipliers (67%) and for the force feedback computation they consist of 36 multipliers (90%) of the 40 available. Thus it is not possible to put both algorithms on the FPGA board. Other investigations pointed out that the low level part of our code uses almost 40% of available area. For the point of view of the overall FPGA area occupation, the occupation is: the forward kinematic (41%), the “virtual spring” control (16%), and the force reflection calculation (68%). Thus only the first two algorithms can coexist in this specific board.

2. Speed: The FPGA uses a clock frequency of 40 MHz. Higher frequency are theoretically available but the complexity of our implementation and the maximum delay introduced limit it to this value. The code that controls the joystick motors runs at 850 KHz, because of limitation on the analog output lines. This guarantees a very stable response of the joystick when the control is completely coded on board, in particular if these values are related to human response time and movement speed. The hardware implementation of the algorithm improves software computation of teleoperation with force feedback. The software implementation was carried out using floating point arithmetic. For example, the fully embedded control, named “virtual springs” runs at 5.7 MHz. For bilateral control, the final computation rate is 220 KHz, which is far above that required for good feedback response, of the order of KHz. Update rates of MHz may not be necessary for performance, but demonstrate the potential of the board to include more complex algorithms without a significant loss of performance.
CONCLUSIONS

In this paper we summarize the main features of a general purpose software architecture for teleoperation, called Penelope, developed at the Robotics laboratory ALTAIR of the University of Verona (Italy). The objective of the architecture is to create an environment for testing various types, topologies and algorithms for force reflection teleoperation, without the need of expensive development tools, and easily supporting reconfiguration and modularity. Additional benefits of this architecture include the possibility of replacing software modules with their equivalent implementation in hardware and, as a consequence, the possibility to improve performance and safety, by significantly increasing the algorithm processing time and by reducing failures due to operating system problems. After a brief summary of significant results in architectures for robotics, we present the structure of the Penelope system, its main classes and the relation among them. Since Penelope is based on the CORBA communication middleware, we ran tests to verify whether a performance decrease was present in the system due to the overhead induced by CORBA. Tests show that the performance decrease was measurable only on older computers, but that the latest hardware generation is fast enough to make up for the increased overhead. To enhance the operator interface of a force feedback teleoperation system, we also ported some of the algorithms for master control to an FPGA board, with the goal of achieving a computation cycle fast enough to allow environment modelling and contact force computation in real time. Although FPGA size has not permitted yet to put all the algorithms in hardware, our preliminary test show that no significant computation errors (round off, truncation, etc) are generated due to the hardware algorithms and that the computation speed show almost a hundred fold increase. From the experience and the tests made, we can conclude that a high performance, high reliability teleoperation architecture can be designed and developed within the resources and capabilities of a university research laboratory. Such a software could form the basis for teleoperation experiments across different laboratories and, once the interface with flight software has been established, it could allow to communicate and interact with experiments in real time. Our plans for future development are focused on improving safety and capabilities of Penelope and to use it in high demand applications, such as surgical robotics and service robot operations, where robotics devices are in close contact with humans.

REFERENCES

8. MIRO, Middleware for autonomos mobile ROBots, http://smart.informatik.uni-klm.de/MIRO/.