DEVICE COMMUNITIES DEVELOPMENT TOOLKIT: AN INTRODUCTION

Riccardo Cassinis
Paolo Meriggi
Department of Electronics for Automation, University of Brescia, Italy
cassinis@ing.unibs.it, meriggi@ing.unibs.it

Andrea Bonarini
Matteo Matteucci
Department of Electronics and Information, Politecnico di Milano, Italy
bonarini@elet.polimi.it, matteucc@elet.polimi.it

Abstract
In this paper we propose the main guidelines of a toolkit designed to allow the several different electronic devices (CPUs, microcontrollers, sensor DSPs, etc.) that compose a mobile robot and its environment to cooperate as members of a community. This idea would allow an easy modularization of complex systems that integrate robots and distributed sensors, taking an effective advantage of the newest data processing and communication technologies. The availability of many computing-capable sensing and actuating devices and of broadband communication channels allows us to find new ways of implementing physically distributed systems that can be used in the control of autonomous robots. Our research is aimed at creating an efficient and reliable layered middleware in the hardware-software structure, in order to allow each device to interact in a transparent way with all the others, on the same machine or elsewhere.

Keywords
Mobile robotics, electronic device communities, multi-agent systems, message oriented middleware.

1. Introduction
There is a general trend in the world of information processing towards distributing computational tasks among several units. The physical location where each part of the computation actually takes place depends nowadays much more on the capabilities of the processor and/or on other opportunity considerations than on the vicinity of the processing unit to the place where data are needed, as it occurred in the past. This allows an enormously greater flexibility in the design of processing systems and of their architectures.
Networking has also become quite an independent issue: most of the communication that takes place among computational units uses the communication system in a completely transparent way, and protocols like TCP/IP make the user totally unaware of the physical means that are being used to transport information.
Such considerations, together with the availability of powerful and efficient wireless data communication means, make it appealing to transfer the concepts of up-to-date computing, such as client-server architectures, applets, servlets, peer-to-peer architectures, etc. to the world of robotics.
Obviously, since the methods that were so far developed are mostly related to other problems than robotics, a new dedicated layer will have to be defined and implemented to allow transparent communication of "robot related" information without losing efficiency. We propose *DCDT* (*Device Community Development Toolkit*), a toolkit to implement such communication layer among different devices making the underlying communication media transparent to the developer.
The next section is devoted to a brief analysis of the changes occurring in the Information and Communication Technology field, and section 3 is particularly focused on the implications in robotics and automation. We will then expose the main theoretical aspects of considering a set of electronic devices as a community in section 4.
The overview of the toolkit's structural aspects is shown in sections 5 and 6, respectively dealing with a general and a detailed description of the toolkit's core.
A description of the ongoing and future work is also included.

2. From Client/Server to Peer to Peer: technology trends and their relevance to robotics
Although being a commonplace, it is rather interesting for our discussion to have a look at the changes that are occurring throughout the whole Information and Communication Technology area (see e.g. [BJO 01]), and then deal with some of the meanings of these changes from a mobile robotics point of view.

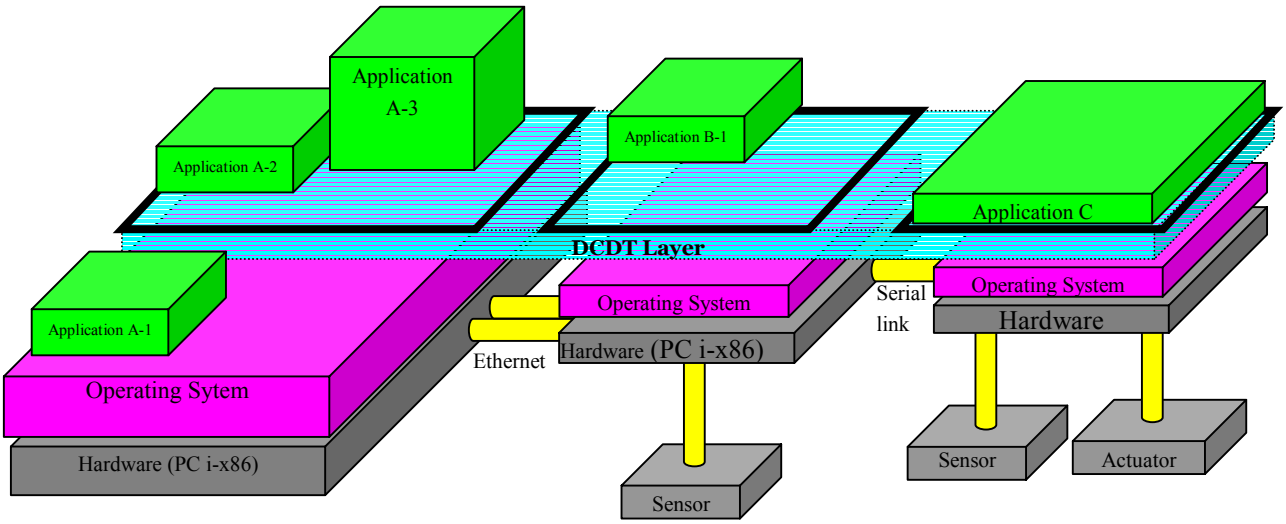

Figure 1 - Example of implementation of the toolkit on three different architectures.

The explosion of computational capabilities (well explained by Moore's law [MOO 65]), the continuous drop of silicon devices size and cost and the growing pervasiveness of the "Internet Paradigm" are undoubtedly changing our everyday habits, leading to what has been called *ubiquitous computing* [WEI 93]. We are rapidly fading from a vertical and strictly hierarchical interaction among computers, to a horizontal, loose *peer-to-peer* model, in which each processing unit becomes a node of a sub-part of the Net [ROM 00].

Similar trends are characterising the automation and industrial robotics progress: the new wave of *smart-sensors* and *smart-actuators* [AKE 00] underlines how, even in the more static industrial environment, the outcome in having more computational power distributed on the field is really remarkable.

Hardwired electronic solutions and handcrafted communication protocols slow serial lines, are leaving their place to flexible MIPS-capable processing units networked through broadband Mega-BPS infrastructures, communicating via well-known standard protocols, both at physical and at logical levels.

Recently proposed communication standards over specific fieldbus systems testify such a trend: DeviceNET [W-09], CANOpen [W-02], Interbus [W-05], Profibus [W-06] and OLE for Process Control (OPC) [W-07] represent many of the efforts the industrial world is undertaking in order to appropriately use the new technological potentials.

The noticeable fact here is that the more we distribute computing power, the higher becomes the level at which each device can interact with the rest of the devices. Each single device (sensor, controlled actuator, controller, CPU, DSP, etc.), that used to be a functional part of a whole, now starts being an active and in some way independent member. In the field of robotics, this applies (or will apply in the next future) even to the most simple and humble devices.

3. Robotics and Automation

Since when the first proposals concerning multi-robot systems were issued, topics such as work sharing, resource allocation, etc. became part of the roboticist's glossary [CAS 86]. This became even more common when multi-sensor systems appeared, and when robot swarms were proposed as a solution to complex robotic problems [SCH 00, DAL 00].

However, most researchers are still considering the sensing system of a robot as a concentration of devices, all physically located in the same place, i.e. on the robot's body. Distributed sensing systems are instead much more interesting in all those cases where robots are to be employed in an environment that, for other reasons, is already equipped with a communication network and with sensors (for example, TV cameras), and when multiple robots exist and co-operate in the same environment.

The latter scheme is particularly important in applications, such as humanitarian de-mining, where it is virtually impossible to fit all necessary sensors on a single vehicle, without making it excessively large and heavy, but where data from multiple sensors are necessary in order to determine whether a mine has been found or not [CAS 00]. Other interesting applications can be done where the robot finds an operating environment that is already equipped, for other reasons, with sensors, communication and computing devices, as it happens in offices, hospitals, etc.

In order to make the system able to effectively use information from a distributed sensing system, several stringent requirements must be met, such as a precise knowledge of the time and of the position of each robot when data were collected, the availability of a common reference system, etc. In most cases, a *real-time* behaviour of the whole system is required.

More generally speaking, considering robots as communities of devices that are not attached to the same

physical body allows many interesting functions to be implemented. This affects sensors (for example, a robot can take advantage of surveillance cameras that observe the environment where it is working as a means for self-localisation), but can also affect processing power, allowing processing units "outside" the robot body to co-operate, offering their services only when and where required.
Given the heterogeneous nature of the robotic systems, most of the distributed frameworks that were proposed for other applications, although very interesting and reliable, exhibit some limitations because they were mainly conceived for homogeneous networks of single CPUs.

4. Communities of devices: *social* aspects

Several authors [ARK 98, MAT 94] have figured out the analogy between robot sets and live beings, from simple cells to swarms of insects, or colonies. According to these theories, we can consider multi-agent systems as sets of some sort.
We propose a more thorough approach: what if the robot itself were seen as a community of devices, deeply connected with the other communities (robots) and the environment in which they are embedded? Although this concept is almost as old as robotics itself, it can get a new significance if seen in the light of the new technologies we are talking about. Such approaches would lead to the study of *communities of communities*, or, better, delocalized communities.
We preferred the term *community* because it refers directly to the *social* aspects of the set. This assumption is extremely important. If among living beings, every social behaviour is intrinsically dependent from the communication, without which there could be no society nor community, and since each mobile robot is a set of devices that includes several processing units, the focus has to be shifted to communication. From this assumption, it immediately follows that considering a group of electronic devices as a community requires a *common communication infrastructure* of some sort.
Among electronic devices, however, we may find a great variety of *languages*, if we consider the way devices interact with each-other: serial and parallel connections, networking physical media, wireless and satellite links, visual and acoustic signalling, etc. It would be really inefficient to force all the communication physical channels to use the same protocol, or to use the same means. This holds true especially in the autonomous robots field, where sensors and actuators may be interconnected in countless different ways.
Since a unique world-wide accepted standard for communication among electronic devices will most likely never exist, our work aims at the realisation of a middleware through which processes running on different devices could interact with each other regardless of the communication channel they are using, providing at the same time a good degree of efficiency in such interactions.
One possible and easy analogy of the approach we are proposing is exactly the way human beings communicate with each other at present days: there are several physical media (Internet, TV, telephone, newspaper, mail), and several "protocols": English, Italian, Chinese etc. Some of these are widely used (e.g. English Spanish, and Chinese), but it would be very ineffective to impose a universal language. Sometimes the problem can be solved using a widely-accepted language (e.g. English on the Internet), but there are cases where the language to be used at one or at both ends of the communication channel is fixed, because changing it would be too expensive or time-consuming. In this case, if an appropriate interpreter is not available, communication is impossible.
Basing our architectural view on what devices exchange with each other and the aim of coping with highly etherogeneous structures are probably the topics that mainly differentiates our approach from other distributed systems commonly used in robotics, such as ETHNOS [PIA 96, PIA 99], Object-Oriented Real-time Framework for Distributed Control Systems [TRA 99], NEXUS [FER 98], OSACA [W-03], ControlShell [W-04], etc.

5. The Device Communities Development Toolkit

The aim of DCDT is to provide an effective tool for building distributed autonomous robot systems. It aims at finding a trade-off among the following, sometimes contrasting, features:

- Supporting easy high level development
- Supporting heterogeneous programming languages
- Supporting heterogeneous hardware
- Supporting rich communication
- Offering a high overall performance

The main idea of the development toolkit is to create or extend an already existing *Message Oriented Middleware* (*MOM*) [W-1] effectively abstracting a multi-channel message transport layer to enable asynchronous exchange of data, event notification, persistence, quality of service, and ease of development.
Component oriented middleware presents peers with a simple model for using remote control and processing logic packaged as components. A peer does not have to worry about the technical complexity of DLLs, remote processes, dynamic memory management, and various layers of communication: this is managed by the middleware. Peers simply request information or the execution of an action, or provide such services in a symmetric and transparent way.
To reach the goals of DCDT a pure component middleware is not enough: we need message reliability, true real-time load balancing, event notification service, persistence, quality of service, cross platform data marshalling, and security support. This can be obtained enhancing the intelligence of the communication layer while preserving the component-based model. In contrast with *remote procedure calls*, the toolkit does not

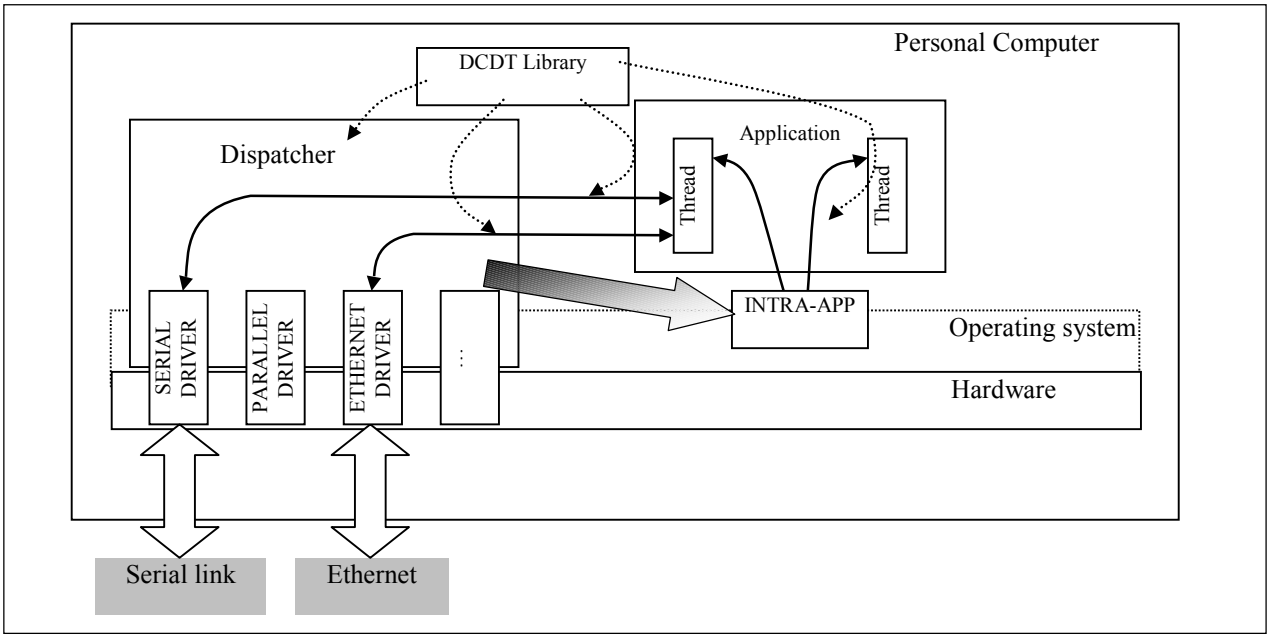

Figure 2 - Example of implementation of the toolkit on three different Architectures.

model messages as method calls, but as events in an event delivery system. Peers send and receive events, or "messages", via the *application programming interface* (API) provided by DCDT. The toolkit enables the development of different kinds of communities depending on their social organization. They may offer either a directory service that lets clients look for other devices that are acting as service providers, or all-purpose "channels" that let a group of clients communicate as peers, or both options at the same time.

DCDT has a *publish/subscribe* interface to enable devices to publish and receive messages they are interested in. This is done by means of a *message bus* that provides a convenient abstraction of the physical transport layer managed by a common interface, implemented through the *dispatcher*. Data coming directly from the sensors or processed by the applications are available to all members of the community, once they accept to share the common layer. Moreover, provided that all the functional constraints (maximum latency, etc.) are respected, all members of the community may have access to the actuators physically connected to one of them.

The *dispatcher* is the real core of multi-channel management, message delivery, subscription propagation, and service providing. Each dispatcher maintains routing tables dynamically built on peer subscriptions that redirect information from producers to interested consumers, avoiding bandwidth consuming broadcasts. It is a task of the dispatcher to select the best communication channel, and the particular protocol to use (i.e. TCP/IP or UDP over Ethernet, shared memory or FIFOs for the applications running on the same machine, etc.) according to predefined efficiency, performance, or reliability criteria. In figure 1 we can see a simple example of a community built through the toolkit middleware. This scheme shows three different processing units: a common desktop PC (A), a small PC-like unit (B) and a microcontroller (C). The first two are linked through a LAN, while the second and the third one are connected via a serial line. Not all the applications running on these three devices have to use the capabilities provided by the toolkit, nor even to deal with the OS or the rest of the world through DCDT. We see, for instance, that on PC1 there are two applications which use the layer (A-2 and A-3), while a third one is simply running on the specific OS of that machine (A-1).

The dispatcher has to deal with all the communication channels it can "see" and that are of interest to any member of the application of the community. When instantiated, it tries to find all the communication channels available to that particular processing unit and other members of the community. This is because it could also be used as a sort of gateway between different communication media, even if on the particular machine on which the dispatcher is running, there are no DCDT applications at all.

When a DCDT application starts running, it communicates to the underlying dispatcher its *needs* and its *capabilities* via a publish/subscribe mechanism. This initial handshaking sequence (that has to be refreshed from time to time) allows the dispatchers to easily select the best way for DCDT applications to interact with each other. Clearly all this raises many networking issues regarding routing and the dynamics involved with the system (e.g., if a device moves from a wireless subnet to another).

In figure 2 we can see an example of a single machine on which two different threads are using the dispatcher, by shared memory or other IPC mechanisms and two of the communication channels available to that processing unit.

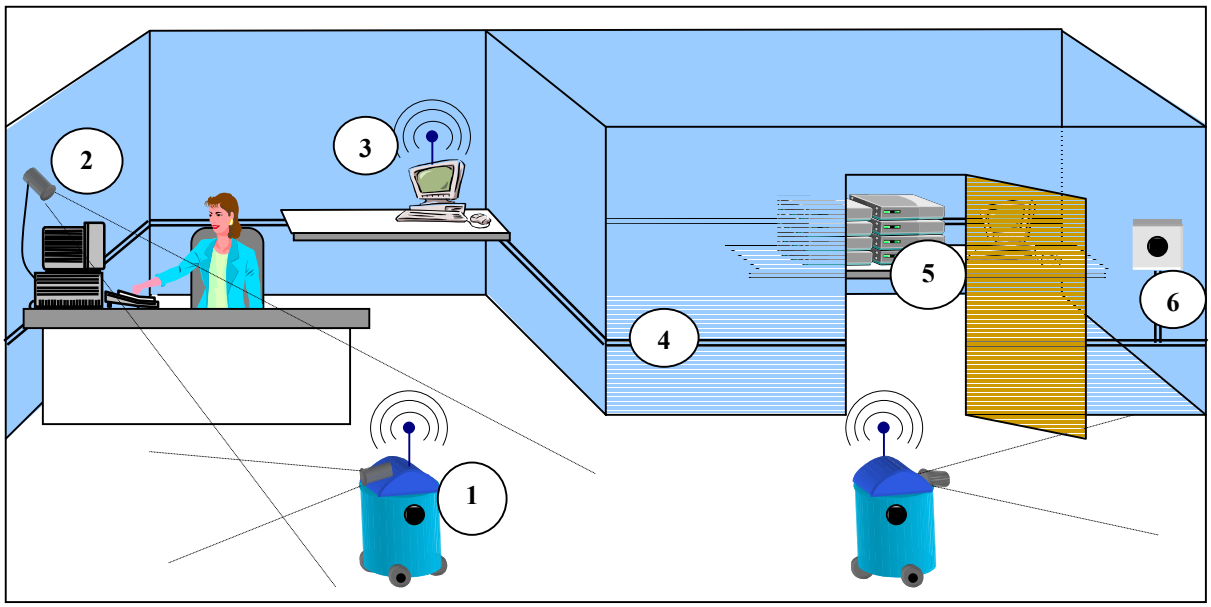

Figure 3 - Possible implementation of the DCDT in a department environment with two mobile robots: (1) Robot, (2) MS Windows station with surveillance webcam , (3) Linux station for robot supervision and gateway between ethernet and wireless network , (4) Ethernet network, (5) Gateway to the Internet (6) Acoustic Sensor Available on the LAN

From the software realisation point of view, we would like to include our effort among those belonging to the Open Robot COntrol Systems (OROCOS [W-08]), both as a way of distributing and developing software. According to this approach the reference component middleware chosen for DCDT is an efficient implementation of CORBA [W-01, BUR 96, PAO 97] specifications allowing different language bindings, free licences, and supporting inter- and intra-application communication.

6. DCDT: technological issues and dispatcher services

The main services provided to developers by DCDT through the dispatcher implementations can be summarised as follows.

Message subscribing and filtering

The API implemented by DCDT allows the creation of peers with the double role of information suppliers and consumers. Peers subscribe to the information they are interested in, and declare which information they are able to produce; this makes it possible to introduce message filtering policies to gain routing optimisation.

Moreover, messages have two special attributes: priority and expiration time. The supplier of an event typically sets these properties. There are many scenarios, especially in robotics, where the consumer's opinion about the relative importance of the event may differ from that of the supplier. Therefore, also consumers are enabled to affect the priority and expiration time of messages. Suppliers and consumers have the ability to query the dispatcher to determine respectively if there is absence of clients subscribing to their specific event information, or if there is no supplier for the information they are interested in.

Quality of service (QoS) and fault tolerance

Robotics applications, and industrial ones in general, require quality of service and reliability in message delivery. DCDT enables peer developers to specify the delivery quality, giving a complete control on dispatcher routing policies:

- *Normal* delivery simply invokes the underlying physical transport layer relying only on the dispatcher routing service;
- *Reliable* delivery keeps trying to deliver the message until the dispatcher receives a positive acknowledgement from the server, or until a time-out or error retry limit is reached;
- *Persistent* delivery stores a message at the dispatcher if the consumer is not available to receive the message.

At the maximum level of QoS reliability, an event is delivered to all subscribed consumers on the message bus. If the connection between the channel and a consumer is lost for any reason and no alternative routes can be found, the channel persistently stores the event information for that consumer. Storage continues until the event reaches its time expiration limit, or until the consumer becomes available again. Upon restart from a failure, the dispatcher automatically re-establishes connections to consumers that were connected at the time the failure occurred.

Moreover, the dispatcher can be configured to occasionally execute a *Normal* delivery as a *Reliable* one, thus checking if the message is correctly delivered. If it is not successfully delivered and no exception is thrown, the dispatcher assumes that the subscription has gone

away and cancels it. If an exception is thrown when an attempt to deliver fails, it assumes that the subscriber is busy and discards the event, but it does not cancel the subscription. The automatic cancellation of dead *Normal* subscriptions provides a cleanup mechanism for subscribers that forgot to unsubscribe.

Real-time delivery scheduling

Real time is an important issue to tackle. If we decide to distribute computational power over several processing units, we may have different real-time *granularities*: encoders and actuators, for instance, must have a very small time granularity, in order to deal with real motions without losing effectiveness, or worse. Instead, central processing units may have a less strict time synchronisation, allowing them to use typical non-deterministic dynamic mechanisms such as multitasking, multithreading, dynamic memory allocation and so on. Nevertheless, it is clearly required that tasks involved in both low-level fast reactive loops and high-level *management* meet their own deadlines without delay. In order to cope with such problem it is required that the transactions between two devices are compatible with the time constrains of the devices themselves, and so communication must be set up at the beginning and periodically rechecked in order to ensure adequate real-time behaviour.

Abstraction: flexibility, portability and scalability vs. efficiency

Applications running on a device of the community may use DCDT in order to communicate to other members of the community, or choose to deal directly with the connected devices through standard or custom methods. DCDT provides a component-oriented abstraction that allows programmers to freely choose the most suitable programming languages and then to bind their code to the communication API. Moreover, DCDT can be used with different architectures: this enables the use of computers as *bridges* or *gateways* between different communication systems (i.e. Wavelan to Ethernet, serial to Ethernet, etc.). Also scalability is obtained through abstraction; in fact, it allows the use of new communication channels and/or protocols as soon as the operating systems are ready to handle them.

7. Ongoing and future work

The work done so far has been mainly devoted to the realisation of the suitable infrastructures, both hardware and software, needed for our purposes: some robots involved in the RoboCup challenge, the robot MARMOT (Mobile Advanced Robot for Multiple Office Tasks) and an initial implementation of DCDT software architecture. In particular the example of Marmot could be helpful in explaining an important issue of DCDT philosophy: since the low level governance of this robot is demanded to a very simple microcontroller (Motorola 68HC11), which has very few if any mathematical capabilities, it could rely on external CPUs to have its trajectory calculations executed abroad, while it continues to control the motion of the robot.

At the present stage of the research, we are completing the DCDT software architecture core prototyping and testing. In fact, although the toolkit itself is conceived to be implemented in many different ways, we chose a few ones as an initial testbed. In particular we are testing DCDT capabilities by means of some mobile robots, equipped with a common CPU and Linux, intelligent sensors (i.e. with processing units) and a microcontroller for the low-level governance of the actuators. All the robots are connected through a wireless LAN. In addition, there are some workstations running Linux and Windows as Operating Systems, connected both to the wireless LAN and to the departmental ethernet backbone.

A possible implementation of the work we are carrying on can be seen in figure 3: an office environment with two robots provided with radio links, several other computers with different operating systems connected through a LAN, and two different “intelligent” sensors (one acoustic and one visual). These sensors could be connected directly to the Ethernet departmental backbone or even to a common office PC, which would become a sort of gateway .

This configuration will be used as a basis for a growing system that will eventually include some “almost sensorless” robots: very simple machines with very small onboard computing power, remotely connected and capable of taking advantage of the whole power of the community.

Moreover, since DCDT is not solely dedicated to mobile robotics, it could be easily extended to other domains, in which there is the need to have different devices interacting with each other through different media. Possible examples could be the growing home and office automation, telepresence, etc.

Investigation in this direction will also have to be done.

8. Conclusions

Beside all the efforts that are being made by many research groups in finding new architectural structures in the mobile robotics field, our contribution consists both of a theoretical approach (considering a set of devices as a community) and of a practical implementation through DCDT.

Indeed we are only at the beginning of the road, but the interdisciplinary view and implications of our work are really interesting in perspective.

In this paper we introduced some of these issues, some other are still to be faced adequately: among all, considering electronic devices as communities, raises serious questions about the relationships involving physical and virtual communities of human beings and of artificial devices.

References

[AKE 00] H.A.Akeel, SW.Holland, “Product and Technology trends for Industrial Robots”, in Proc. IEEE Int. Conf. On Robotics and Automation, San Francisco, California, April 2000, pp. 696-700.

[ARK 98] R. Arkin, “Behavior-based Robotics”, MIT Press, 1998.

[BJO 01] N.Björkman, Y.Jiang, T.Lundberg, A.Latour-Henner, A.Doria, “The movement from Monoliths to Component-Based Network Elements”, IEEE Communications Magazine, January 2001, pp. 86-93.

[BUR 96] R.L. Burchard, Feddema, “Generic Robotic and Motion Control API Based on GISC-Kit Technology and CORBA Communications”, in Proc. IEEE Int. Conf. Robotics and Automation, Apr. 1996, pp. 712-717.

[CAS 86] R. Cassinis, "An Application of Automatic Resource Sharing to Robot Programming", in Proc. III International Symposium of Robotics Research, Faugeras, Giralt (Ed.), MIT Press, Boston, USA, 1986

[CAS 00] R. Cassinis, "Landmines Detection Methods Using Swarms of Simple Robots", in Proc. International Conference on Intelligent Autonomous Systems 6, E. Pagello (Ed.), Venice, Italy, 2000

[DAL 00] B.Dalton and K.Taylor, “Distributed Robotics over the Internet”, IEEE Robotics and Automation Magazine, June 2000, pp.22-27

[FER 98] J.A.Fernandez, J.Gonzalez, “NEXUS: A Flexible, Efficient and Robust Framework for Integrating Software Components of a Robotic System”, in Proc. of the 1998 IEEE ICRA, Leuven Belgium, 1998.

[MAT 94] M. Mataric, “Interaction and Intelligent Behavior”, Technical Report AIM-1495, MIT AI Lab, 1994.

[MOO 65] G.E. Moore, “Electronics”, Volume 38 Number 8, 1965, pp. 114-117

[PAO 97] C. Paolini and M. Vuskovic, “Integration of a robotics laboratory using CORBA”, in Proc. IEEE Int. Conf. Systems, Man and Cybernetics, Vol. 2, 1997, pp 1775-1780.

[PIA 96] M.Piaggio, A.Sgorgbissa and R.Zaccaria, “A Distributed Architecture for Autonomous Robots”, Proc. IEEE Int. Conf. in Engineering of Complex Systems, Montreal, Canada, 1996.

[PIA 99] M.Piaggio, A.Sgorgbissa and R.Zaccaria, “ETHNOS-II A Programming Environment for Distributed Multiple Robotic Systems“, in IEEE Proc. Hawaii Int. Conf. on System Sciences, Hawaii, 1999

[ROM 00] G.C. Roman, G. Picco, and A Murphy: “Software Engineering for Mobility: a Roadmap”, in A. Finkelstein, ed., The future of Software Engineering, ACM Press, 2000.

[SCH 00] D.Schulz, W.Burgard, D.Fox, S.Thrun, A.B. Cremers, “Web Interfaces for Mobile Robots in Public Places”, IEEE Robotics and Automation Magazine, March 2000, pp. 48-56

[TRA 99] A.Traub and R.D. Schraft, “An Object Oriented Realtime Framework for Distributed Control Systems”, in Proc. IEEE Int. Conf. On Robotics and Automation, Detroit, Michigan, May 1999, pp. 3115-3121

[WEI 93] M. Weiser, “Some computer science problems in ubiquitous computing”, Communication of the ACM, July 1993.

[W-01] http://www.omg.org/corba/

[W-02] http://can-cia.de

[W-03] http://www.osaca.org

[W-04] http://www.rti.com/products/controlshell/CS.html

[W-05] http://ibsclub.com/index.htm

[W-06] http://www.profibus.com

[W-07] http://www.opcfoundation.org

[W-08] http://www.orocos.org

[W-09] http://www.infoside.de/infida/wissen_devicenet.htm