ETHNOS-II
A Programming Environment for Distributed Multiple Robotic Systems

Maurizio Piaggio, Antonio Sgorbissa, Renato Zaccaria
(piaggio@dist.unige.it, sgorbissa@dist.unige.it, zaccaria@dist.unige.it)
DIST Laboratorium, University of Genova, Via Opera Pia 13, Genova, Italy

Abstract
Most robot software architectures focus on the problem of imitating human intelligence and thus typically refer to a single robot perceiving, navigating and acting in the environment. However, the rapid progress of communication technology has modified this reference scenario, offering the possibility of “distributing” the intelligent activity on a network of robots, computers and other general sensing and actuating devices. This allows the robot to merge with the environment it operates in and, moreover, different robots may co-operate as a single entity in order to carry out a specific task more efficiently. This paper tackles this “extended” problem, presenting ETHNOS-II, a programming environment for the design of a system composed of different robots integrated with the environment they operate in. ETHNOS-II provides support from two main point of views: from the software engineering perspective it provides support for platform independence, software integration and re-use, computation distribution; from the runtime perspective it provides support for real-time execution and event handling, inter-robot communication, and intra-robot resource allocation.

1. Introduction
The problem of designing an autonomous robot capable of operating intelligently in a civil environment, only partially known and in which objects and people move around, is still an open research issue. This is witnessed by the large amount of work in literature addressing this topic: in particular, in recent years, many software architectures for the planning and control of mobile robots have been proposed, and different workshops or journal special issues organised [1] [2].

These architectures mostly focus on the problem of imitating human intelligence (whether throughout the interaction of reactive behaviours, integrating reactivity and deliberation, exploiting diagrammatic forms of reasoning, etc.) and thus typically refer to a single robot perceiving, navigating and acting in the environment (i.e. in a hospitals, museums, etc.) as a person would (examples are AuRA [3], 3T [4], etc.). However, the rapid progress of communication technology has modified these reference scenarios, offering the possibility of “distributing” the intelligent activity on a network of robots, computers and other general sensing and actuating devices [5]. This not only results in greater available computational power (for example, in the case of mobile robots, processing is no longer limited to the on-board unit) but it allows the robot to merge with the environment it operates in. In suitable intelligent buildings a mobile robot may open doors, turn on/off lights or even avoid obstacles relying not only on its sensors and actuators but on the interaction with other robotic entities (lights, doors, etc.). Moreover, different robots may co-operate as a single entity in order to carry out a specific task more efficiently.

This paper tackles this “extended” problem, presenting ETHNOS-II, a programming environment for the design of a system composed of different robots integrated with the environment they operate in. ETHNOS-II provides support from two main point of views:

• from the software engineering perspective it provides support for platform independence, software integration and re-use, computation distribution;
• from the runtime perspective it provides support for real-time execution and event handling, inter-robot communication, and intra-robot resource allocation.

This paper will discuss these properties in detail. In the next section we will describe our motivations deriving from our previous work in this field: the initial ETHNOS [6] environment and its limitations will be briefly dealt with. The third section will present a new
reference cognitive model for mobile robots to which the new ETHNOS-II environment refers. The fourth section focuses on the characteristics of ETHNOS-II itself and the fifth section will provide some experimental results validating our claims. Finally conclusions follow.

2. The Original ETHNOS Environment and its Limitations

ETHNOS (Expert Tribe in a Hybrid Network Operating System\(^1\)) [6] is an operating system designed to support the implementation and execution of a distributed multi-agent cognitive model [7] in which a deliberative symbolic component running on a remote workstation communicated asynchronously, throughout a radio link, with a reactive and partially reactive component running on-board of the mobile robot. The system was based entirely on the concept of expert, a concurrent agent responsible for a specific deliberative or reactive behaviour. Its main goal was to encourage the system developer to de-couple the different experts in execution, to reach, as close as possible, the limit situation in which the single expert is not aware of what and how many other experts are running. In this way an expert could have been added, removed or modified at run-time without altering the other components of the system.

This property, in our opinion, is essential in intelligent systems in which different skills have to be dynamically selected, depending on the current task(s) to be carried out, from the complete set of robot expertise and flexibly combined to achieve the desired behaviour. Thus flexibility and adaptability are key issues in this sense. Moreover, in ETHNOS, the strong independence of the single experts also intended to facilitate the integration of different techniques.

Expert de-coupling was achieved by eliminating any static communication link. Experts communicated, with the EIEP (Expert Information Exchange Protocol) [8], essentially an efficient implementation of a message blackboard (based on a publish/subscribe technique) in a network distributed environment. However, in our experience with the use of ETHNOS, despite its many advantages which lead to the development of successful autonomous robots [9], and despite different, blackboard based, robotic systems have been presented in literature [10][11], we have encountered two main problems:

- Experts often need to communicate implicitly throughout analogical, image based, representations [12]. However, the exchange of a representation throughout the use of messages is intrinsically forbidden in practice by its memory dimensions, whereas using a single expert to mediate the access to the shared representation has many disadvantages in terms of efficiency, priority handling and flexibility;
- the EIEP message based technique is exclusively asynchronous but, especially in the interaction with sensors and actuators or when strict real-time behaviour is required, some information can be better exchanged synchronously.

Moreover, the system was designed to allow the expert mobility in the computer network. Thus there was no distinction (apart from evident timing and availability considerations) from the programming point of view between experts running on-board and experts running remotely: communication was dealt transparently by the system. This property proved to be powerful in single robot applications but introduced a significant limitation in multi-robot scenarios, not allowing to distinguish experts running on one robot from experts running on another robot.

These and other considerations in autonomous robot architectural requirements (both from a cognitive and an engineering perspective) lead us to the development of a new cognitive model HEIR and to an extension of ETHNOS to provide the missing functionality and to overcome the limitations found\(^2\).

3. HEIR Cognitive Architecture

This section presents a cognitive architecture for autonomous robots, HEIR (Hybrid Experts in Intelligent

\(^1\)In ETHNOS the expression “expert tribe” signified an analogy between a robotic system and a tribe composed of many experts (concurrent agents) in different fields (perception, actuation, planning, etc.). The term “hybrid” indicated the support to the integration of deliberative experts with experts responsible for reactive and partially reactive behaviours; the term “network” referred to the possibility of distributing components (groups of experts) on a computer network.

\(^2\) It is worth mentioning that also the original EIEP introduced two communication methods: shared memory access and direct service access responsible respectively for sharing analogical representations and providing synchronous expert communication. However they were introduced only as exceptional methods (to be used only if strictly necessary), operating in contrast to the de-coupling property on which ETHNOS was founded.
Robots) [13], and its extension to the multi-robot scenario. The structure of the cognitive architecture is depicted in fig. 1. It is organised non-hierarchically into three components characterised by the type of knowledge they deal with: a symbolic component (S), handling a declarative explicit propositional formalism, a diagrammatic component (D), dealing with analogical, iconic representations, and a reactive behaviour based component (R). The architecture, is based on experts (depicted as circles), classified depending on the component they belong to: symbolic(S), diagrammatic(D) or reactive(R). Each group corresponds to a different set of computational tasks, distinguished depending on: the type of cognitive activity carried out, timing constraints, type of data managed, duration, etc.

This distinction helps to identify specific architectural requirements and operating system support for the different components. They fall into three main categories:

- **Representation requirements.** Symbolic experts are based on a common symbolic knowledge base (KB), the global encyclopaedic information on the world they operate on in order to carry out their activities. Diagrammatic experts exploit diagrammatic, iconic representations [12] (depicted as boxes labelled D1, D2, etc.). The difference in this case is both on the representation itself (closer to metric data coming from sensors) and on the number of icons present. In fact, whereas the KB is one and common to all symbolic experts, the iconic representations are many and possibly different. Moreover each icon can be managed or simply accessed by one or more experts. Reactive experts do not have any significant representation at all (they differ slightly from Brook’s behaviours [14] because they may have a small internal state).

- **Communication requirements.** In HEIR there are two main communication mechanisms: one related to representation: symbolic experts communicate
throughout the KB and diagrammatic experts throughout the iconic descriptions; another based on messages conveying information of any type: symbolic, diagrammatic or simply metric. The latter has two purposes: one related to the reactive component in which these messages may be exchanged both synchronously and asynchronously to allow reactive expert real time interaction, another related in asynchronous inter-component communication to allow the integration of the different representation paradigms.

- **Execution requirements.** In this perspective, timing requirements are more strict in the reactive component and progressively relax in the diagrammatic and symbolic components. In fact reactive experts are subject to real time constraints deriving both from their interaction with sensors and actuators, and from the necessity of handling real world environments that are both dynamic and unpredictable. Diagrammatic experts still have to respond efficiently: from the execution point of view it is only necessary to synchronise their access to the share analogical representations in order to avoid inconsistent states. Symbolic expert operations must be serialised because of the nature of symbolic inferential processed. However this is not a problem because they normally have longer, non real-time, unpredictable duration.

The cognitive architecture above can be extended, in a straightforward way, to take into account a multi-robot scenario or a scenario in which the robot communicates with external devices. Different configurations can be envisaged depending on the tasks that need to be carried out: figure 2 illustrates a configuration of three robots in which the different symbolic components interact to allow the robot to co-operate.

In figure 3 there is only one symbolic component, shared among all the robots and in figure 4 the robots also communicate with external devices returning or receiving instructions under the form of symbolic informations (i.e. switches to turn on/off lights) or analogical data (i.e. closed circuit cameras also performing some analogical operations on the images acquired, thermometers returning the temperature of the environment, etc.)

4. ETHNOS-II Distributed Environment

The ETHNOS-II programming environment is based on the cognitive architecture presented in the preceding section. It is composed of:

1. **ETHNOS-II,** a dedicated distributed real-time operating system, from which the overall environment takes its name, supporting HEIR's representation, communication, and execution requirements,
2. a dedicated network protocol designed for both the single robot and the multi-robot environment, specifically designed for noisy wireless communication,
3. an object oriented Application Programming Interface (API) based on the C++ language (and a subset based on Java),
4. a set of additional development tools (a robot simulator, a Java-applet template, etc.)

The first two points regard what we have called the runtime perspective whereas the last two regard the software engineering perspective. Let's examine them in detail.

4.1 ETHNOS-II Operating System

The ETHNOS-II operating system is designed as an extension of POSIX compliant operating systems (in particular the Linux Real-Time OS has been used). It performs different functions:

- it supports the creation of, and access to shared representations (the KB or the different iconic instances). Representations are characterised by a type to which they belong (i.e. top-view bitmap, navigation potential field, etc.). When an expert creates a representation it specifies the type to which it belongs; in alternative, if, interrogating the system, it discovers that a representation of the same type has already been created, it connects as a creator to the existing one. Users (experts that only use the representation information) "connect", indirectly throughout the OS, referring to the type only and not to the representation instance directly. This avoids rigid coupling because an expert that needs to use a specific representation only focuses on the representation type and not on the expert or experts providing it. The distinction between creators and users is necessary to let the system know (and as a consequence the connected experts) if a representation is still kept updated (i.e. at least one creator is present).
- It allows the concurrent execution of different experts transparently and efficiently implemented as real-time threads, and their transparent synchronisation when accessing the above shared resources (multiple readers and writers problem).
- It supports both asynchronous and synchronous communication. In the former, analogously to the original EIEP, information is dispatched using a publish/subscribe technique to avoid static expert coupling. An expert subscribes to the message types it desired to receive (i.e. positioning data, collision detection, etc.) and, from then onwards, receives any message of that type published by any expert in the system. In the latter the experts declare, during their initialisation phase, the synchronous services they provide, again specifying their type. Analogously, experts advance the system requests about specific service types. It will be the OS that will provide this connection at run time throughout indirect function calls. Note that this allows the system to automatically dynamically reconfigure without any activity by the experts involved: for example lets consider the situation in figure 5a for a mobile robot. There are five experts in the system: a motion controller that sends commands to the actuators, a sensor expert that perceives information about the environment surrounding the robot, a map handler that builds a representation of the world the robot moves in, a positioner that tracks the robot position based on odometry and finally a navigator that, on the basis of the robot position and map status informs the motion controller on the navigation strategy. The links in the figure are thus self explaining. Lets imagine that the sensor system suddenly no longer function correctly but a more precise positioning system based on active beacons is available. The architecture can easily adapt to the new configuration by substituting the positioner with an active beacons positioner, the navigator with a blind navigator and terminating both the sensor expert and map handler that are no longer of use. Figure 5b illustrates the new configuration. It is important to notice that the motion controller was not at all modified nor informed of the environmental changes.

![Figure 5 - Reconfigurability example](image)
• It supports transparent asynchronous communication in the computer network, within the single robot configuration. Experts running on a particular machine do not have to be aware of the location of the message receivers (not even of their existence), whether on the same machine or on another one connected in the network. Clearly this is different if inter-robot communication is involved. In this case it is possible to distinguish internal messages (meaning messages to be distributed within the machines implementing a single robot instance) and external messages (meaning messages to be sent from a robot to another).
• It allows experts to suspend, waiting for the occurrence of specific events (reception of particular message types or even combinations of them) to avoid wasting computational time. Moreover it allows each expert to specify a desired period of execution which the Posix real time scheduler will transparently be instructed to respect.

4.2 EEUDP Network Communication Protocol

In ETHNOS-II network communication is often wireless (i.e. radio link, wavelan, etc.). Thus, because of interference or because the robot may have moved to a blind zone, transmission packets are more frequently lost. In this context, both TCP-IP and UDP-IP based communication cannot be used: the former because it is intrinsically not efficient in a noisy environment; the latter because it does not guarantee the arrival of a message, nor any information on whether it has arrived or not. For this reason we have designed a protocol for this type of applications, called EEUDP (Ethnos Extended UDP) because, based on the UDP, it extends it with the necessary properties.

The EEUDP allows the transmission of messages with different priorities. The minimum priority corresponds to the basic UDP (there is no guarantee on the message arrival) and should be used for data of little importance or data that is frequently updated (for example the robot position in the environment that is periodically published). The maximum priority is similar to TCP because the message is sent until its reception is acknowledged. However, it differs because it does not guarantee that the order of arrival of the different messages is identical to the order in which they have been sent (irrelevant in ETHNOS-II applications because every message is independent of the others), which is the cause of the major TCP overhead. Different in-between priorities allow the re-transmission of a message until its reception is acknowledged for different time periods (i.e. 5 ms, 10 ms, etc.).

4.3 Application Programming Interface

In addition to the cognitive architectural requirements, ETHNOS-II has been designed to facilitate the development of complex distributed real-time applications even by non highly specialised users. Object oriented technology has been used to simplify the programming of new experts, their integration and the re-use of existing code. A generic expert is in fact provided by a suitable API - application programming interface library. This expert encapsulates all the properties and available services the system provides, allowing an implicit integration in the architecture, without any in-depth knowledge of how the services are implemented. New expert need simply to inherit this base expert class and redefine only their specific and distinguishing behaviour. Thus the derived expert will add its personal public methods that characterise the external interface of the object. Necessarily it will specify the operations that must be carried out when the expert is activated or deactivated as well as the main operation that it must execute.

The API has been written for the C++ language on which the entire system is based. However for communication to external devices (such as user consoles) a subset of the API (containing the necessary communication services) has also been written of the Java language allowing an easy development of Ethnos integrated Java applets. This allows a user to communicate with the robot even from common network browsers.

Other, still generic, classes are also provided for symbolic, diagrammatic and reactive experts. These classes, as depicted in Figure 6 belong to the generic base class but include the specific intra-class properties. Also in this case the user is not required to have any knowledge of the implementation of real-time services.

![Expert class hierarchy](image)

Figure 6 - Expert class hierarchy
providing synchronised access to a representation or synchronous communication.

Clearly, object oriented design can be extended to the entire application, with different expert class hierarchies created by the system users. It is important to notice that, this not only facilitates code re-use and integration, but it also does not alter the basic architectural mechanisms. In fact, the experts, despite their possible extensive inheritance, are finally executed in a flat, non hierarchical architecture and they can therefore still respect their real-time constraints.

Concerning platform independence, the developer benefits from the expert de-coupling property which allows to easily substitute an expert with one or more providing similar functionality. In particular the experts more closely related to the electronic and mechanical parts of the robot (sensors, actuators, kinematics, etc.) can be substituted to execute the application (with the remaining part left unaltered) on different robots or on a simulator.

4.4 Additional Development Tools

Additional tools have also been developed to facilitate the development of complex robot applications. In particular a multi-robot simulator has been designed, together with a suitable interface library which allows its transparent connection to an ETHNOS-II based system. Other tools consist in: Java applet templates for user interface development and Interface libraries with a terminological system (CLASSIC [15]).

![Figure 7 - Experimental Architecture](image-url)
5. Experimental Results

An experimental architecture developed using the ETHNOS-II programming environment is depicted in figure 7. The horizontal dotted lines separate the different components: from top to bottom, symbolic, diagrammatic and reactive.

The reactive component is composed of five experts: the sonar perceiver, the motion controller, the positioner, the emergency controller, the trajectory generator. The first three experts are connected to sensors and actuators; the last two are more robot independent. Their global behaviour is to navigate the robot in a given position in the environment, executing smooth "blind" trajectories (in the sense that they do not take into account the presence of obstacles in the path except for emergency halting operations).

The diagrammatic component builds a map of the environment based on sonar sensor data and an abstract potential field. The navigator virtually navigates in this field, "reaching" intermediate target positions that guarantee obstacle avoidance. The reactive level will be instructed to physically head smoothly towards these positions. At the same time the roadmapper extracts a symbolic graph description on the environment from the analogical information in the map.

The symbolic component uses the information in the roadmap and combines it with the map knowledge of the environment, obtaining a more complete description which classifies the different regions of space (i.e. room centres, narrow passages, turning points, etc.). This representation is used by the planner to solve complex navigation tasks.

Clearly this is only a brief description since a longer one would be outside the scope of this paper (for more detailed information refer to [16]). Its aim, however, is to illustrate the properties of ETHNOS-II in the engineering of a complex distributed computer system, exploiting different paradigms of representation, different forms of communication and real-time execution constraints.

From the execution perspective the system automatically takes care of the execution of the different experts with their different timing constraints: strict real-time for the low-level reactive experts, real-time in the diagrammatic component and with more relaxed timing in the symbolic component. Communication is transparently taken care of: in our system the two lower components were implemented on board of the robot, whereas the symbolic component executed on a remote workstation connected throughout a 9600 baud radio link. At the same time, the access to the different representations (KB, MAP, APF) is transparently synchronised (in the sense that an expert is not allowed to read the contents of a representation while another expert is changing it), avoiding inconsistent states.

From the software engineering perspective it provides a framework for the expert development that allows a faster integration of experts produced by different programmers. At the same time it relieves the programmers from difficult, highly specialised tasks, such as low-level real-time concurrent programming and network communication.

The structure of the reference architecture HEIR also facilitates the re-use of the same system on different robot platforms. For example, the architecture described in figure 7 has been implemented originally on a TRC Labmate and successively easily ported to a Pioneer Robot and to a robot called Snoopy, partly developed in our laboratory. Figure 8 shows the different robots used, all equipped, with differences in computational power and number and sensor displacement, with an on-board pentium processor and ultrasonic proximity sensors.

Figure 8 - Labmate, Pioneer, Snoopy Robots.
The overhead due to the ETHNOS-II system has also been experimentally evaluated and did not result significant. For example, the context switch time per expert (CSTE) expresses the overhead of the system (ETHNOS-II + underlying Linux RT) in switching from the execution of an expert to another. Figure 4 illustrates the value of CSTE parameter, on a Pentium Intel 133Mhz, relatively to the number of experts in execution.

![Figure 4 Plot of CSTE(µs) versus number of experts](image)

The time taken is 17 µs per expert, with an overhead of approximately 10 µs with respect to the typical Linux RT thread context switch time (7 µs). If we consider that the application is composed of 7 experts, operating at a frequency of 40 Hz, at each cycle, the global expert computational time will be 25 ms and the time used by the OS in context switches will approximately be 0.11 ms. Thus, in this case, the system uses globally 0.44 % of the computational resources for the expert scheduling.

ETHNOS and ETHNOS-II based robotic systems such as the one illustrated in the previous section have been developed and implemented in our laboratories and they have been successfully used both in industrial set-ups for pallet transportation (Ceramiche Lonardo Factory, Imola (FO), 1995/1996) and in museums or shows for public entertainment (Exhibition Sculture Gutenberghiane e Manifesto dell’Antilibro, Museo della Scienza e della Tecnica, Milano, March-April 1997, Fair Salone Formula, Magazzini del Cotone Congress Center, Genova, April 8-12 1997, Fair Salone Formula, Magazzini del Cotone Congress Center, Genova, March 1998), meeting so far approximately more than 3,500 people. A photo taken during one of such public appearances is shown in figure 9.

In the different applications the robots were capable of performing complex navigation tasks in large crowded environments, avoiding both static objects and moving people as well as vocally interacting with them. In our laboratory the robot was connected, using a remote control, to the automatic sliding door and it was therefore capable of opening and closing it whenever it was necessary to perform the given mission. ETHNOS-II has also been used during the RoboCup competition in Paris 1998, in the middle-size robot league, for inter-robot communication and decentralised co-ordination within the Italian Azzurra Robot Team (ART) which arrived to the quarter-finals of the competition.

![Figure 9 - Labmate at Fair Salone Formula](image)

6. Conclusions

This paper presented a distributed real-time programming environment for the development of autonomous mobile robots integrated in the environment they operate in. Its properties from the execution and software engineering point of view were discussed and some examples of its applications described. The system, in both its initial version and in this second one, has been now used for more than three years in our laboratory and its success witnessed by the different public appearances. Additional work is being carried out at different levels: in particular further studies on the interaction between the symbolic, analogical and procedural representation paradigms and on analogical forms of reasoning. Moreover, in addition the ad hoc communication protocol, we are studying a system scheduler (in substitution to the standard real-time Posix scheduler) specifically designed and suited for intelligent autonomous vehicles.
References


