What Planner for Ambient Intelligence Applications?

Francesco Amigoni, *Associate Member, IEEE*, Nicola Gatti, *Member, IEEE*, Carlo Pinciroli, and Manuel Roveri

***Abstract*—The development of ambient intelligence (AmI) applications that effectively adapt to the needs of the users and environments requires, among other things, the presence of planning mechanisms for goal-oriented behavior. Planning is intended as the ability of an AmI system to build a course of actions that, when carried out by the devices in the environment, achieve a given goal. The problem of planning in AmI has not yet been adequately explored in literature. In this paper, we propose a planning system for AmI applications, based on the hierarchical task network (HTN) approach and called distributed hierarchical task network (D-HTN), able to find courses of actions to address given goals. The plans produced by D-HTN are flexibly tailored to exploit the capabilities of the devices currently available in the environment in the best way. We discuss both the architecture and the implementation of D-HTN. Moreover, we present some of the experimental results that validated the proposed planner in a realistic application scenario in which an AmI system monitors and answers the needs of a diabetic patient.**

***Index Terms*—Building-management systems, cooperative systems, distributed computing, planning.**

I. Introduction

As implicitly appears in the definition of ambient intelligent (AmI) proposed by the European Commission Information Society Technologies Advisory Group—"a potential future in which we will be surrounded by intelligent objects and in which the environment will recognize the presence of persons and will respond to it in an undetectable manner" [1]—the very general idea of AmI involves, among others, concepts related to distributed intelligence and autonomy. Recently, some authors (for example, [2]) stressed the need to combine distributed intelligence paradigms within the architectural levels identifiable in AmI systems; namely, ubiquitous computing devices [3] (in particular pervasive computing devices [4]), ubiquitous wireless communication [5], and intelligent multimodal interfaces [6]. Distributed goal-oriented behavior enables the system to take the initiative when appropriate and to improve the capabilities of the distributed reactive behavior (that simply reacts to changes in the environment) often implemented in AmI systems [7]. As a consequence, goal-oriented behavior extends the range of problems that can be tackled by an AmI system.

In this paper, we focus on the *planning* ability, namely on the most natural way to provide goal-oriented behavior to AmI systems. An AmI system that plans is able to find a course of action that, when executed, achieves a desired effect (for example, the satisfaction of a user's need), given the current state of the environment and the repertoire of actions the devices composing the system can perform. While most of the research in AmI concentrates on obtaining information about a user's state and behavior [8]–[10], the planning system proposed in this paper addresses the topic of using the information coming from sensors to perform actions useful for the user [11].

Despite its potential importance, planning in AmI has not received adequate attention so far. Some attempts to incorporate planning in AmI systems have addressed the *recognition of the user's plans* in order to proactively assist and facilitate the user to perform the activities of the plans [12], [13]. In this paper, we take a different perspective, presenting a planning system that *builds plans* according to the capabilities of available devices that perform actions to satisfy the user's need. Note that, in the first case, the plans are usually selected from a library [14]; while, in the second case, our system automatically builds plans to reach high-level goals provided as input. Moreover, in the first case, the system has no control over the user who executes a plan, while in the second case, our system has some sort of control over the devices that execute a plan. In this paper, we will refer to this last conception of planning that involves building plans whose actions are performed by AmI devices.

In our opinion, the difficulty in developing a planner of this kind might be traced back to the following two peculiar characteristics of AmI systems.

1) It is widely recognized that the power, processing, and communication limitations [15] of the the distributed devices composing an AmI system prevent to consider them *a priori* as able to perform complex computations, in particular, those required by planning tasks. For example, a cell phone could not be able to autonomously plan to call a doctor given that other devices detected that a user in the environment is ill.
2) Because of the very nature of AmI systems, the number and the types of the distributed devices that are connected to an AmI system are not known in advance and continuously change during the operations of the system. For example, when a person with a smart cell phone enters a room equipped with an AmI system, the cell phone, when properly connected to the network of other devices, is temporarily part of the system.

These two features make it difficult to directly employ traditional planning techniques in AmI. Feature 1 pulls toward *centralized* planning [16], while feature 2 pushes toward *distributed* planning [16]. Given the above dichotomy, this paper aims at shedding some light on the answer to the question: what is the most suitable planner for AmI applications? To the best of our knowledge we are not aware of any planner that successfully addresses the above question at the light of features 1 and 2. We propose a planner, called distributed hierarchical task network (D-HTN) that combines both centralized and distributed features

Manuscript received October 15, 2003; revised April 1, 2004 and June 15, 2004. This paper was recommended by Guest Editor G. L. Foresti.

The authors are with the Dipartimento di Elettronica e Informazione, the Politecnico di Milano, Milan 20133, Italy (e-mail: amigoni@elet.polimi.it).

Digital Object Identifier 10.1109/TSMCA.2004.838465

and that represents an innovative and original contribution to the research in AmI. D-HTN performs a centralized plan-building activity that is tailored on the capabilities of the distributed devices that are currently connected to the system.

The autonomy and the goal-oriented behavior promoted by the planning ability of AmI devices acting in dynamic environments is envisaged, although in a primitive and limited form, in the area of *context awareness* [17], that aims at developing devices that behave according to the context. These devices are able to automatically tune some configuration parameters in response to specific contexts and can undertake different courses of actions in different contexts (thus exhibiting an active behavior). Research in context awareness has been oriented almost exclusively to single devices. When multiple devices are involved, we can distinguish between two kinds of context awareness: context awareness toward the user (i.e., the context is what the user is currently engaged in and what the user is trying to do [18]) and context awareness toward the system (i.e., the context is the information about the devices currently connected to the system). The first kind of context awareness is usually addressed by the designer of the AmI application, while the second kind of context awareness can be addressed to a large extent by the computational infrastructure on which the AmI system is implemented. The planning system proposed in this paper focuses on context awareness toward the system. To coordinate the operations of different devices, the interaction patterns among the devices (e.g., the coordinated sequences of actions they perform) are usually predetermined by the designer. For example, in [19], an event starts the execution of a chain of actions performed by different devices; the chain of actions has been statically predefined by the designer. We overcome these limitations by providing a planner for AmI applications that iteratively decomposes a task in simpler subtasks until they can be mapped on the actions that the (currently connected) devices can perform. This means that the interaction patterns are not statically predetermined but are automatically built according to the initial goal and the devices currently connected to the system.

The main original contributions of this paper are: 1) the characterization of the planning problem within AmI; 2) the proposal of a planner for AmI applications that combines centralized and distributed features and that is based on the theoretical and experimental background of HTN planning [20]; and 3) the ability of the proposed planner to adapt the planning process and its results to the capabilities of the devices currently connected to the AmI system.

In this paper, we adopt the multiagent paradigm [21], [22] for designing and implementing AmI systems (following, for example, [18] and [23]). As outlined in [24], the multiagent approach finds a somehow "natural" application within AmI, since it enables the development of systems in which the intelligent and autonomous functions [25] are exhibited by both the individual devices (namely, at the individual level) and by their interaction in a network of devices (namely, at the social level). An early proposal to use the multiagent paradigm in smart homes appeared in [26]; however, this proposal did not take into account the aspects related to the coordination and cooperation among the agents. Two reasons for our adoption of the multiagent approach in AmI are that the theoretical and architectural discussions on distributed planners are conveniently casted in a multiagent perspective and that the existing frameworks for distributed systems, such as JINI [27], can be fruitfully employed in implementing multiagent systems.

This paper is organized as follows. Section II presents some realistic application scenarios that help in motivating the use of planners in AmI applications and to which we will refer through the paper. In Section III, we discuss in detail the problems arising when developing a planner for AmI applications and, starting from features 1 and 2 discussed above, we review the state of the art in planning. Section IV presents the architecture and some theoretical properties of the proposed D-HTN planner for AmI applications. Implementation details are illustrated in Section V. In Section VI, we experimentally validate the planner in our application scenario. Section VII concludes the paper.

II. Application Scenarios

Imagine a room equipped with smart devices forming an AmI system. Imagine now that a person suffering heart diseases [28] and wearing other smart devices suddenly becomes seriously ill while being in the room. If the smart devices (agents, in our perspective) in the room can interact with the smart devices (agents) carried by the person, these latter could communicate the alarming state of affairs to the former so that they can manage such a critical state. The AmI system can decide immediately to call a medical specialist, sending information about patient's vital signals [29]. The AmI system can use a cell phone, an e-mail system, or a fax according to the devices currently available in the room. If the specialist does not respond in a short time and the devices evaluate the health state excessively critical, the AmI system can make an emergency call to the closest hospital and request an ambulance, employing the most efficient communication means. The monitoring of people suffering heart failure can be properly addressed by using a dynamic collection of agents organized in an AmI system. Such a system requires several devices that, in general, can vary according to the particular environment (e.g., hospital, home, and outdoor), and an intensive collaboration among them. Information about the patient is retrieved from scales measuring patient's weight, blood pressure measurement devices, electric heart activity monitoring devices, and physical activity monitoring devices. These data are used to determine the level of the patient's physical activity, to set the drug doses the patient has to assume, and to alert physicians and to autonomously defibrillate if a critical state occurs. Note that a planning system for assisting patients suffering heart failure that is tailored on the available devices can be used in different environments.

This motivating AmI scenario is currently being defined in cooperation with experts in cardiac diseases. The scenario, although extremely realistic, is not yet completely implementable in reality; however, the technological building blocks are already available or being actively developed. To make the discussion more concrete and to partially relate to our previous work [30], we will refer, in the rest of this paper, to another similar application scenario. We consider a diabetic patient equipped with on-body monitoring devices who is in a room equipped with

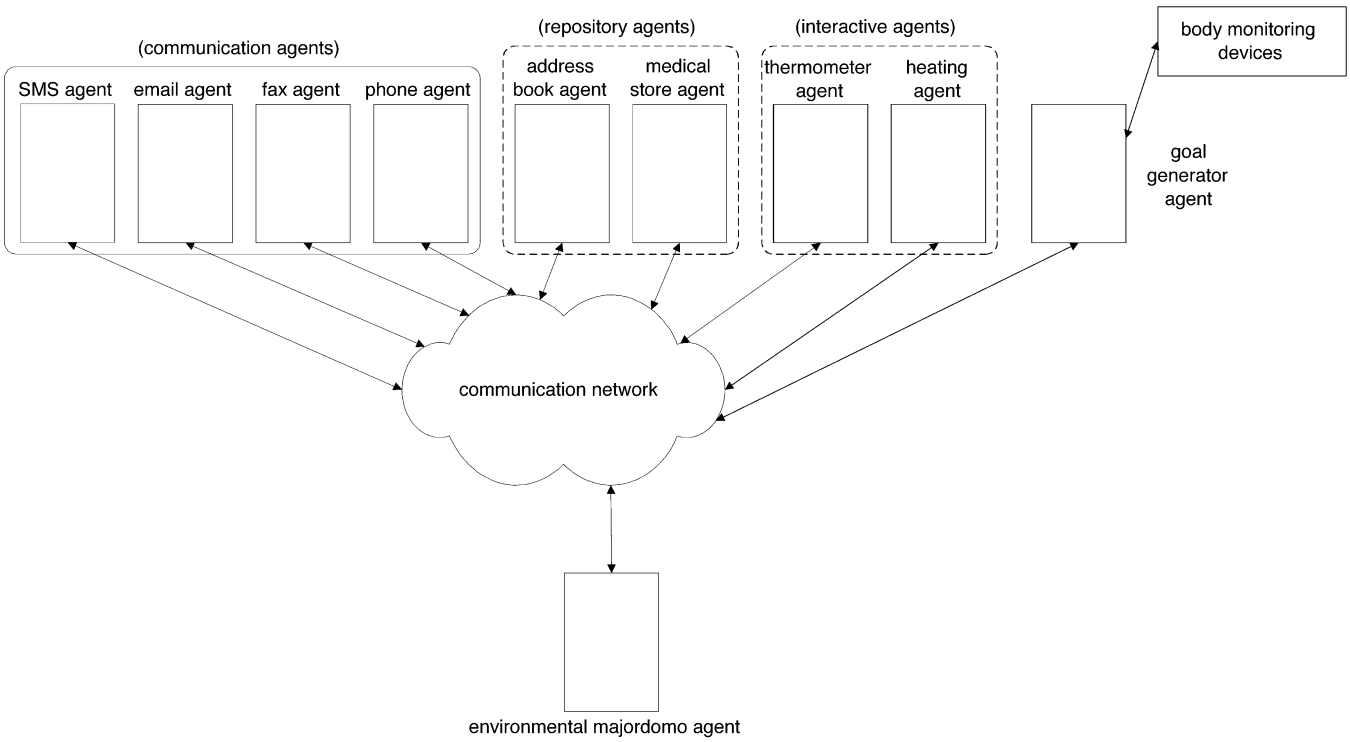


Fig. 1. Architecture of the AmI system we implemented for the application scenario.

AmI devices. (Note that those related to diabetes patients are among the most studied applications of medical telemonitoring [31].) In the diabetes case, the patient seldom becomes suddenly ill because the monitoring devices can usually detect potential alarming situations some time before they appear. However, the planning abilities required to the AmI system are roughly the same in both the heart disease and the diabetes scenarios. We work on a simulation of the diabetes scenario that abstracts from many technological issues of the devices (these issues, although of paramount importance, see for example [32], are outside the main scope of this paper and are partially addressed in Section V). Hence, for our purposes, all the AmI devices are embedded in software agents that are integrated in a multiagent system. This architecture is further discussed in Section V.

In our experimental scenario we implemented the following agents (Fig. 1). The agent that simulates the monitoring devices on the diabetic patient is called *goal generator agent* because in our application scenario it is the source of goals that the planning system attempts to achieve. The goal generator agent stands for any device or user that can generate a goal for the AmI system. Note that this agent might represent (i.e., be connected to) a whole multiagent system devoted to regulate and monitor physiological processes, such as the one we developed in [30]. Hence, the goal generator agent provides the input to our planner in terms of high-level goals to be reached. The agents that populate the room in our scenario are conceptually organized in three main classes: *communication agents*, *repository agents*, and *interactive agents*. The communication agents include the SMS *agent*, the *email agent*, the *fax agent*, and the *phone agent* for sending and receiving SMS, e-mails, faxes, and phone calls, respectively. The repository agents are the *address book agent*, a database of contacts, and the *medical store agent*, a database of medicines currently present in the environment. The interactive agents provide the sensors and the actuators to interact with the environment; they include the *thermometer agent*, a temperature sensor, and the *heating agent* that can change the temperature in the environment. All these agents (the goal generator agent and those equipping the room) are supervised and coordinated by the *environmental majordomo agent*. Our experimental system manages both the agents that represent devices that are physically and permanently part of the room (e.g., the heating agent) and the agents that represent mobile devices that are transiently part of the room (e.g., the phone agent could be a cell phone carried by a person walking through the room). The dynamic management of devices is based on a discovery mechanism (provided by JINI in our implementation) that allows the AmI system to incorporate mobile devices and these devices to effectively use their surroundings [18].

In the following of this paper we discuss the topic of goal-oriented behavior within the AmI multiagent system just described, which provides a realistic testbed for validating our planner.

III. PLANNING AND AMI

In this section, we address in more detail the problem of planning (we refer to planning for system devices in the sense discussed in Section I) in AmI applications by relating our approach with literature on planning.

In order to exhibit goal-oriented behavior in the real world, a system has to generate a plan and to coordinate the execution of the plan. In this paper, we are concerned only with the first aspect, called *planning*. Basically, a planning algorithm has three inputs [33]: a description of the *world*, a description of the *goal*, and a description of the *capabilities* in form of possible actions

that can be performed. The planning algorithm's output is a sequence of actions such that, when they are executed in a domain satisfying the initial state description, the goal will be achieved.

When we try to apply the above definition to an AmI scenario, the fundamental aspect of *distribution* [34] calls for consideration. From Section I, we already know that AmI systems are characterized by features 1 and 2. Let us consider their implications on planning in more detail. Feature 2 concerns the fact that in an AmI system the skills to perceive the environment and to perform the actions are distributed over the agents and this pulls toward the distribution of the planning process. However, according to feature 1, in AmI some agents could take no responsibility in building the plan since their limitations in processing and communication; and this pushes toward the centralization of the planning process. Therefore, an AmI planner cannot be fully distributed and the planning process should be centralized in an agent that has enough processing power. In conclusion, in an AmI system we need *a centralized planner that manages distributed capabilities*.

In the literature on centralized planners [33], [35], the hierarchical task network (HTN) planners (originally introduced in [36] and [37]) are generally considered the most suitable for real-world applications, as long as a single planning agent is concerned. For this reason, in this paper we propose (a modification of) an HTN planner for application in AmI scenarios. These planners are based on the concept of *task network* that is represented as (the following description is based on [20]):

$$[(n_1 : \alpha_1), (n_2 : \alpha_2), \ldots, (n_m : \alpha_m), \phi]$$

where

- α_i are *tasks*, either *primitive* (that can be directly executed by an agent) or *nonprimitive* (that must be further decomposed);
- n_i are labels to distinguish different occurrences of the same task;
- ϕ is a Boolean formula representing the constraints on the tasks, such as variable bindings constraints [e.g., $(v = v')$], ordering constraints [e.g., $(n \prec n')$, with the meaning that n must be executed before n'], and state constraints [e.g., (n, l, n'), with the meaning that l must be true immediately after n, immediately before n', and in all states between n and n'].

A task network can be represented by a graph. For example, the task network:

$$\begin{aligned}&[(n_1 : \texttt{CreateRequestText}(g_1, t_1)),\\ &\quad (n_2 : \texttt{SearchEmailAddress}(g_1, a_1)),\\ &\quad (n_3 : \texttt{SendEmail}(t_1, a_1)), (n_1 \prec n_3) \wedge (n_2 \prec n_3) \wedge\\ &\quad (n_1, \texttt{RequestText}(t_1), n_3) \wedge (n_2, \texttt{EmailAddress}(a_1), n_3)]\end{aligned}$$

is represented by the graph in Fig. 2. The intended meaning is that, in order to request a good g_1 by e-mail, we first have to create the `RequestText` t_1 and look for the `EmailAddress` a_1 of a supplier of g_1, then we have to `SendEmail` with content t_1 to a_1. Note that the language used for HTN planning is an extension of a first-order language.

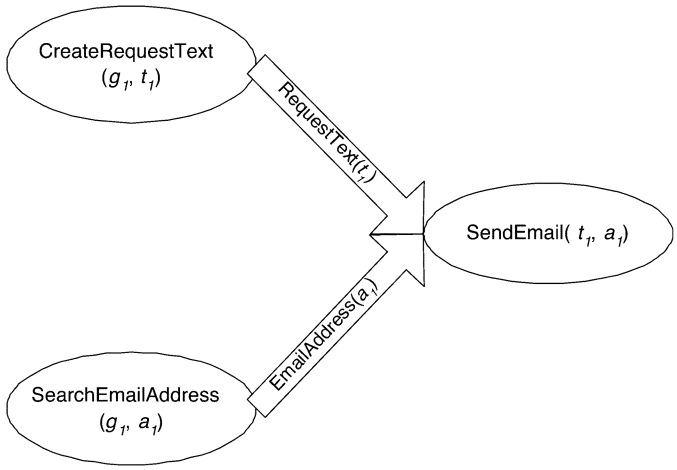


Fig. 2. Graph representing a task network.

HTN planning starts with an initial task network $\mathcal{D}$ representing the problem (the goal) and with a set M of methods or *decompositions*. Each decomposition is a pair $m = (t, d)$, where t is a nonprimitive task and d is a task network; m says that a way to achieve t is to perform the tasks in d. Then, HTN planning proceeds by finding a nonprimitive task t from the current task network $\mathcal{D}$ and a method $m = (t', d')$ in M such that t' unifies with t and by replacing t with d' in $\mathcal{D}$ (after appropriate substitution, see [20]). When only primitive tasks are left in $\mathcal{D}$, a plan for the original problem can be found. A *plan* is a sequence of ground primitive tasks (see [20] for details). This pure HTN planning process can be refined to make it more efficient by introducing backtracking, critic functions, and other technicalities [37]. The most known centralized HTN planner is probably the Sacerdoti's NOAH [37].

In order to cope with feature 2, we consider the distributed versions of HTN planners. The literature on distributed planning usually refers to multiagent planning (since [38]) and classifies distributed planners in two classes [34]. The first class is cooperative distributed planning (CDP). Its main purpose is the creation of a coherent (sometimes optimal) plan. CDP is typically employed with agents that have common objectives and representations and that exchange information about their partial plans, which are iteratively refined and revised until they fit together well. The second class, negotiated distributed planning (NDP), is centered around the idea that the purpose of each agent is to negotiate over planned activities not to form good collective plans, but rather to ensure that its own local objectives will be reached by its own plan, when viewed in a global context (see, for example, [39]–[42]). Therefore, in NDP there is no cooperation to create a joint plan as in CDP, but every agent tries to convince the other agents to accommodate its own preferences.

Given that AmI agents are usually assumed to be cooperative [18], CDP approaches are appropriate for AmI applications. It turns out that most of the works in CDP are extensions of the HTN planning we have illustrated before (notable exceptions are the partial global planning approach [43] and the DIPART platform [44]). This kind of distributed HTN planning operates through successive refinements of the planning decisions as more information about the plans of the other agents becomes

available. A distributed HTN approach appears appropriate for AmI applications because it naturally supports heterogeneous agents and knowledge exchange among them. One of the most successful attempts in distributing HTN planning is Corkill's distributed version of the centralized NOAH planner [distributed NOAH (D-NOAH)] [45]. D-NOAH is based on the same plan representation and the same planning procedure of the classical centralized NOAH, but exploits multiple and distributed centers of planning control (agents, in our perspective). Each agent performs planning, as in NOAH, through the expansion of nonprimitive tasks by finding applicable decompositions that achieve them. In D-NOAH, the task network for plan representation is extended to include special nodes used as placeholders for other agents' plan and to support special primitive operations for the synchronization of the multiagent execution. Tasks are allocated to the agents by the user who specifies which agent solves which top-level goals. Communications and coordination are natively supported by extending the critic functions for detection of the conflicts that are present in NOAH. Note that in D-NOAH, no agent knows all the possible decompositions and, thus, the knowledge is distributed. However, D-NOAH is somehow too powerful for our purposes since we do not need a planner able to manage different centers of planning control (recall feature 1). Hence, we developed our D-HTN planner as a simplification of D-NOAH (or, equivalently, as a complication of NOAH), in order to have a single agent performing the planning process and a communication mechanism for exchanging knowledge about decompositions. The main problem to be addressed is the high dynamism required by AmI systems (recall that agents continuously join and leave the system): D-NOAH does not provide any support in this sense, in fact the number and the types of the distributed planners are static and fixed in advance. Finally, we note that there are several hybrid variants that combine HTN planning with other planning approaches; we discuss them in Section VII.

IV. D-HTN Planner

Generally speaking, the D-HTN planner proposed in this paper is a centralized planning system in which the knowledge about the decompositions is distributed over the agents. The agents, except a single planning agent called *planner*, are not required to take part in the planning activity, they are only required to provide their decompositions. In this section, we give a general description of D-HTN, while implementation details are reported in Section V.

A. General Assumptions

Before presenting the details of the algorithms, it is worth giving an overview of the general assumptions we make. D-HTN is a multiagent planning system in which, given a goal, a number of agents cooperate to allow the planner to build a plan. We assume that the agents and the planner can perform the following activities.

- Each agent keeps a local data structure called *plan library*, which stores all the decompositions it knows. The decompositions in the plan library of an agent have been defined by the designer during the installation of the agent and are peculiar for each agent. In particular, they include decompositions that involve only primitive tasks for the agent, since they are directly executed by the agent without being further expanded.
- The planner is the only entity supposed to have the computational power to generate a plan, the other agents are only requested to communicate decompositions (and to execute functions in the environment).
- By means of a communication mechanism based on message passing, 1) the planner can ask the currently connected agents to send their available decompositions for a given task and 2) the agents can send to the planner the requested decompositions. Note that this communication mechanism is the only way in which D-HTN manages distributed knowledge about the decompositions.

The decompositions $m = (t', d')$ we consider are such that d' can be a *conditional plan* [46, Chapter 12] that uses sensing tasks to select a branch in the plan. In this way, conditional plans can handle bounded indeterminacy. In particular, in d' the **if-then** and **while** constructs control the execution of the tasks. For instance, a task can be executed only if a condition is satisfied. Examples are shown in Section VI.

Each decomposition has associated three numerical indexes (not included in the original formulation of NOAH planner) that are associated to the *performance*, the *cost*, and the *probability of success* for the tasks in the decomposition. The first one measures the expected effectiveness of the decomposition, the second one measures the expected resource consumption for performing the tasks in the decomposition, and the third one measures the expected likeliness that no error occurs when the tasks in the decomposition are performed. For example, the index values (expressed in a scale from 0 to 1000) for the following decomposition that makes a request by phone:

$$\begin{aligned}&(\texttt{Request}(g_1), [(n_1 : \texttt{CreateRequestMessage}(g_1, m_1)),\\ &\quad (n_2 : \texttt{SearchPhoneNumber}(g_1, p_1)),\\ &\quad (n_3 : \texttt{MakeCall}(m_1, p_1)), (n_1 \prec n_3) \wedge\\ &\quad (n_2 \prec n_3) \wedge (n_1, \texttt{RequestMessage}(m_1), n_3)\\ &\quad \wedge (n_2, \texttt{PhoneNumber}(p_1), n_3)])\end{aligned}$$

could be 800, 500, and 500, respectively. On the other hand, the index values for the following decomposition that makes a request by fax:

$$\begin{aligned}&(\texttt{Request}(g_1), [(n_1 : \texttt{CreateRequestText}(g_1, t_1)),\\ &\quad (n_2 : \texttt{SearchPhoneNumber}(g_1, p_1)),\\ &\quad (n_3 : \texttt{SendFax}(t_1, p_1)), (n_1 \prec n_3) \wedge\\ &\quad (n_2 \prec n_3) \wedge (n_1, \texttt{RequestText}(t_1), n_3)\\ &\quad \wedge (n_2, \texttt{PhoneNumber}(p_1), n_3)])\end{aligned}$$

could be 200, 500, and 500, respectively. These index values model the fact that making a phone call is a more effective way to make a request than sending a fax; moreover, the cost and the probability of success (for completing the call and sending the fax) are equal. The values are chosen (and possibly modified) by the designer of the system. This poses a problem when different designers are involved; a solution might be the adoption

of a general shared set of rules for writing decompositions (further details are given in Section VI). The possibility offered by D-HTN planner to define different (redundant) decompositions for the same task improves the robustness of the planner, especially when it acts in dynamic environments. However, conflicts among decompositions for the same task may arise. The three indexes we defined provide a criterion to heuristically select a "good" decomposition when more are applicable. Note that other indexes can be added to the three presented above; for example, we can consider also the estimated time required to carry out the tasks of a decomposition. However, in this paper we only consider the performance, cost, and probability of success indexes because they are enough to manage the decompositions of our application scenarios. Moreover, although we did not implement this possibility, the values of the indexes could be dynamically updated according to the success of previous plans.

Finally, for simplicity and without any loss of generality, the planner does not show any other ability except that of planning.

B. D-HTN Algorithms

D-HTN is composed of a set of distributed algorithms that are executed concurrently by the planner and by the agents. Algorithm 1 presents an overview of the D-HTN algorithm executed by the planning agent. The main data structure to represent the plan that is being formed is a task network $\mathcal{D}$. $\mathcal{D}$ is initialized with the initial task to be solved (i.e., the goal to be reached). The D-HTN planner produces a final plan $\mathcal{D}$ composed only of primitive tasks that can be executed by the agents.

Algorithm 1 D-HTN algorithm for the planner
1: $\mathcal{D}$ = initial task
2: **while** $\mathcal{D}$ contains nonprimitive tasks **do**
3: choose a nonprimitive task t from $\mathcal{D}$
4: `populate` $M(t)$, by requesting the currently connected agents to send the decompositions $m = (t', d')$ such that t' unifies with t and by collecting these decompositions
5: choose a decomposition $m = (t', d')$ from $M(t)$
6: **if** t is primitive for the agent a proposing m **then** bind a to t and remove t from the nonprimitive tasks
8: **end if**
9: replace t with d' in $\mathcal{D}$
10: **end while**

Let us analyze the algorithm in more detail. Step 3 is quite trivial since the order in which we choose nonprimitive tasks for decomposition does not influence the solution eventually found. In Step 5, a decomposition is chosen according to a criterion that depends on the index values of the proposed decompositions (set according to the specific application). For example, in our application scenario, we used the criterion that selects the decomposition (giving precedence to decompositions composed only of primitive tasks in order to favor the termination of the algorithm) with the highest value of the performance index. More sophisticated criteria could be devised (see, for example [25]).

Note that the dynamic collection of decompositions (during Step 4) is a conceptually simple activity that nevertheless has a significant impact in AmI applications; it is an original contribution of this paper (it is not part of any HTN algorithm we are aware of).

In developing D-HTN, we defined a "clean" pattern for message exchanging between the planner and the other agents. In our system, the massages are only exchanged between the planner and the other agents. The planner requests all the agents to suggest possible decompositions for the task at hand, by broadcasting a message including the task to be decomposed. Each agent, if it knows decompositions for that task, informs the planner by sending the known decompositions. We note that, in this paper, we are not interested in the coordination of the execution of the primitive tasks of a final plan. Indeed, we use a trivial but effective mechanism: execution is coordinated by a single agent (see Section VI).

Step 4 of Algorithm 1 populates the set of decompositions $M(t)$ for a task t. Execution of Step 4 is synchronized with the execution of Algorithm 2 by the agents. Implementation of Algorithms 1 and 2 in our application scenario is discussed in Section V.

Algorithm 2 D-HTN algorithm for the agents
1: **while** the agent is active **do**
2: wait for a message from the planner
3: **if** the message is a request of decompositions for a nonprimitive task t **then**
4: send to the planner the decompositions $m = (t', d')$ in the plan library such that t' unifies with t
5: **end if**
6: **end while**

C. Some Theoretical Results

We now turn the attention to some theoretical results about the proposed D-HTN planner. We preliminarily note that, when communication is assumed to be reliable and conditional plans are not allowed in decompositions, D-HTN planner reduces to the general HTN planner defined in [20] (proof is trivial). Hence, under the above restrictive assumptions, all the theoretical results presented in [20] hold also for the D-HTN planner. Some of them are summarized below.

We call PLANEXISTENCE the problem: given a set of decompositions, a set of primitive tasks, an initial state, and a goal, is there a plan that reaches the goal? The most significant decidability results follow.

- In the most general case, without restrictions on nonprimitive tasks, PLANEXISTENCE is semidecidable.
- If the decompositions are acyclic (i.e., any task can be expanded up to only a finite depth), PLANEXISTENCE is decidable.
- If all the task networks in the decompositions are totally ordered (i.e., the tasks need to be achieved serially), then PLANEXISTENCE is decidable.

The most significant complexity results follow.

- In general, PLANEXISTENCE has exponential complexity both in space and in time.

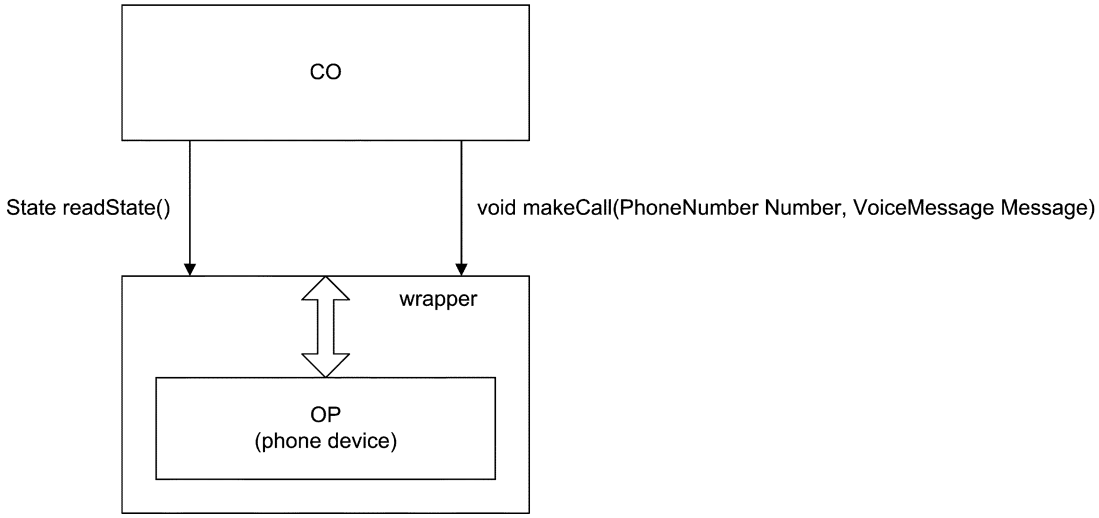


Fig. 3. Wrapper that makes the functions of a phone device available as JAVA methods to the CO to form a phone agent.

- If 1) all the task networks in the decompositions contain at most one nonprimitive task and that nonprimitive task is ordered with respect to all other tasks in the network and 2) the set of decompositions is known in advance (i.e., no agent can enter or leave the AmI system during the planning process), then PLANEXISTENCE has polynomial space complexity.

Note that, although some of the above worst-case theoretical results are negative, they do not prevent the successful use of HTN planning in practical applications [20]. Despite these preliminary results, a complete theoretical analysis of the proposed D-HTN planner is a quite challenging tasks that is beyond the scope of this paper and that has not been yet addressed.

V. IMPLEMENTATION ISSUES

In this section, we present the implementation of the D-HTN planner and the AmI system we developed to test it in the application scenario of Section II. In particular, we discuss how the general requirements listed in Sections II and IV have been implemented using specific technologies. (For a much more detailed description of the implementation issues of D-HTN planner, please refer to [47].)

A. Implementation of the AmI System of Our Application Scenario

The first implementation issue we addressed relates to the potential heterogeneity of the devices composing the AmI system. They could exhibit different hardware and software implementations and communication protocols. In order to cope with this problem, each agent is architecturally arranged in a pair of semiagents: the cooperative semiagent (CO) and the operative semiagent (OP), as shown in Fig. 3. (In [48], we discuss the same idea applied to robotic agents.) The CO accomplishes cooperative tasks, providing the connection to and the disconnection from the AmI system and the communication ability to exchange messages with the other agents. These abilities are exploited in event management (the agent can generate an event, register itself to particular events, and receive the notification that a registered-to event happened) and in the goal management (the agent can generate goals, propose decompositions for a task, carry out an action to execute a plan, and update its plan library). Implementation of event management is covered in [49], while goal management is the topic of this paper. Each CO is a software component implemented in JAVA by exploiting JINI [27] as middleware, which, in turn, is based on RMI [50]. Hence, all the interactions among agents are based on the remote procedure call paradigm. In particular, the messages exchanged by the agents during planning (see previous section) are embedded in the parameters of the remote procedure calls.

The OP is the operative and specific part of the agent. Basically, the OPs are the devices scattered in the environment. Their specific functions can be computational or interactive with the physical world. An OP can be implemented with different architectures and by different programming languages. Thus, usually it is necessary to introduce an additional component commonly employed in multiagent systems, called *wrapper*, that maps the functions that the OP can perform to JAVA methods accessible from the CO (Fig. 3) and that facilitates the reuse of the devices in different applications. By using the wrapper, we allow the remote execution of the CO with respect to the OP since they communicate via RMI. This is useful because usually the CO requires more computational power than the OP. The functions performed by the OP constitute the basis for the primitive tasks of the agent. Given our architectural solution, the integration of real devices (new OP parts) will affect only the wrappers. This shows that our CO-OP modular architecture makes it easy to evolve from prototypical simulated systems to real ones.

To make the discussion more concrete, we briefly describe the implementation of the wrapper we are developing (in cooperation with a Swiss building automation company) for OP devices that are connected by means of the LONWORKS system for building automation [51]. Fig. 4 shows that the wrapper is common to a number of OP devices that are connected to a LONWORKS bus. Such devices include, for example, switches and presence sensors. The JAVA component of the wrapper

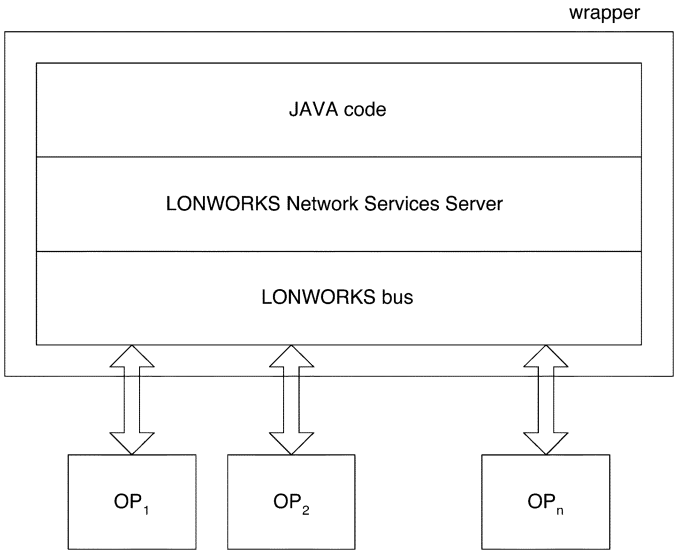


Fig. 4. Wrapper for LONWORKS devices.

can access the variables that characterize the devices (in order to read or write them) by means of the services provided by the LONWORKS Network Services Server. This component provides also a publish-subscribe management of events, such as the modification of a variable value. The JAVA component of the wrapper can also remotely communicate, via RMI, with the CO of the agent. The JAVA component of the wrapper receives (from the CO) the requests to execute operations and translates them in corresponding requests for the OP devices. Note that the LONWORKS Network Services Server and the JAVA components can be run on a computer that is remotely connected to the OP devices by means of the LONWORKS bus. In this way, it is possible to partially overcome the processing and communication limitations of these devices and to integrate them in a coherent agent-based cooperation framework. The communication technologies used in this case are summarized in Fig. 5.

The management and supervision of our agent network is performed by the environmental majordomo agent. It keeps information about the dynamic registration of the agents by embedding the JINI lookup service. The COs of the agents can automatically discover the presence of the environmental majordomo and join the AmI system by using, respectively, the `discovery` and `join` protocols provided by JINI. The adoption of a fixed single center, such as the environmental majordomo, to manage the agent network reduces the communication between the agents, according to the limited communication resources available to AmI agents. In addition, as remarked in [52], such approach faces the potential failure of the interagent communication, due to their transition and rapid moving, by imposing an implicit communication mechanism (via the environmental majordomo).

B. Implementation of the D-HTN Planner

The planning agent described in the previous section is embedded in the environmental majordomo. Note that the environmental majordomo can also perform operative functions and can have its own plan library. In particular, it can answer its own requests for decompositions. This is simply a consequence of our modular design of agent architecture. The D-HTN algorithm for the planner (Algorithm 1) is implemented extending `Thread` in order to allow the environmental majordomo (and the AmI system) to concurrently build different plans for multiple goals.

In our implementation, every agent (actually, every CO) can remotely invoke methods of other agents dynamically connected via JINI. According to that, the environmental majordomo requests decompositions for a task t by remotely invoking the method `requestDecompositions` with a parameter describing t. Then, each agent can reply to such request by remotely invoking the method `proposeDecompositions` with, as a parameter, a list of decompositions for t (those present in the plan library of the agent). The situation is depicted in Fig. 6.

More precisely, `populate` $M(t)$ is called by the planner to request the decompositions (see Step 4 of Algorithm 1) and is detailed in Algorithm 3. The timeout of Step 7 of Algorithm 3 helps to reduce the negative effects on the performances when agents cannot answer promptly due, for example, to communication problems.

Algorithm 3 `populate` $M(t)$
1: $M(t) =$ empty
2: $A =$ set of currently connected agents {obtained from the JINI lookup service}
3: $A_A = \text{empty}$ {A_A is the subset of currently connected agents that already answered the request}
4: **for** each agent $a \in A$ **do**
5: call a.`requestDecompositions`(t)
6: **end for**
7: wait until $A_A \equiv A$ or a timeout has expired
8: **return** $M(t)$

The method `requestDecompositions` (t), outlined as Algorithm 4, is implemented by every agent. The agent returns $N(t) = \emptyset$ when it has not any knowledge about how to decompose t and $N(t) = \{m_1, m_2, \ldots\}$ when it knows how to decompose t. In this latter case, if there exists a $m_i = (t'_i, d'_i)$ such that t' unifies with t and $d'_i = [(n'_i, t'_i)]$ (i.e., d'_i is a task network with a single task), then t is a primitive task for the agent (the agent can perform t by executing a primitive operation associated with t'_i). In order to make the call of `requestDecompositions` not blocking and to allow the planner to send out all requests before answers of the agents, `requestDecompositions` first generates a thread that executes Algorithm 4 and then ends immediately. Note that `requestDecompositions` is the basic functionality that must be provided by an agent to take part to the planning activity: the implementation of `requestDecompositions` is the minimum processing and communication requirement that must be satisfied by an AmI agent of our system.

Algorithm 4 `requestDecompositions` (t)
1: $N(t) = \text{empty}$ {$N(t)$ is the set of decompositions for t the agent knows}
2: **for** each decomposition $m = (t', d')$ belonging to the plan library

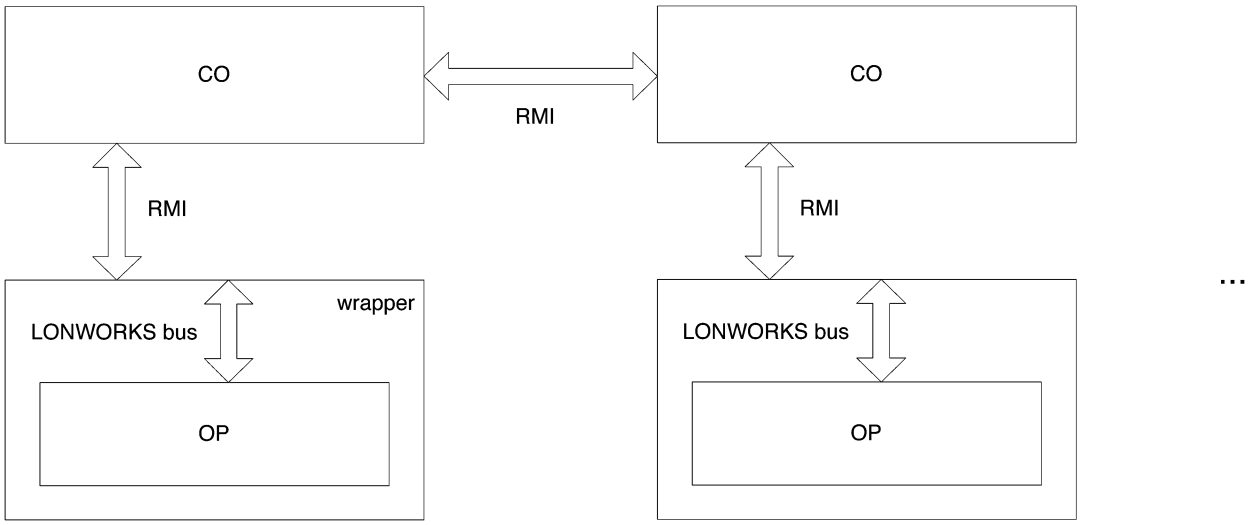


Fig. 5. Communication technologies.

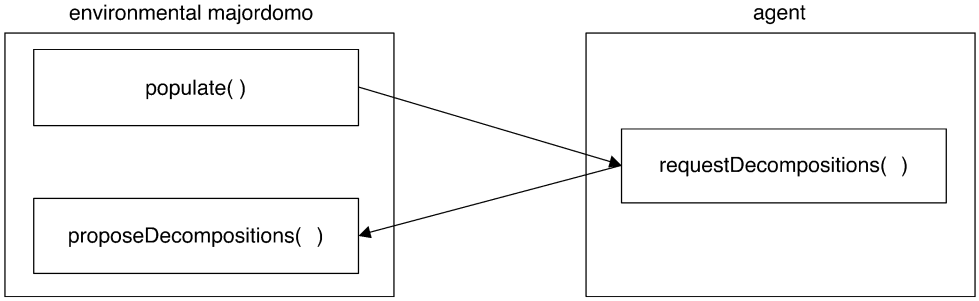


Fig. 6. Remote procedure calls for exchanging decompositions.

do
3: **if** t' unifies with t **then**
4: $N(t) = N(t) \cup \{m\}$
5: **end if**
6: **end for**
7: call e.`proposeDecompositions`$(N(t))$ {e is the environmental majorodomo}

Each agent can propose decompositions by remotely calling (within its instance of `requestDecompositions`) the method `proposeDecompositions` of the planner, shown in Algorithm 5. For simplicity, in Algorithm 5 we do not show the filtering that eliminates the duplicate decompositions and the decompositions conflicting with other tasks already in the plan.

Algorithm 5 `proposeDecompositions` $(N(t))$
1: $A_A = A_A \cup \{a\}$ {a is the agent calling the function}
2: **for** each decomposition $m = (t', d') \in N(t)$ **do**
3: $M(t) = M(t) \cup \{m\}$
4: **end for**

In our implementation, the goal generator agent is the only one that generates goals and sends them to the planner. However, this is a limitation imposed by our application scenario described in Section II and not by the D-HTN planner that can manage goals coming from any agent. The goal generator agent provides a graphical interface (Fig. 7) monitoring in real time the planner behavior: the number of agents registered in the JINI lookup service, the timeout for receiving decompositions (this can be set by the user), the current task to be decomposed, the agents that received the task decomposition request, and, finally, the agents that proposed decompositions for the task.

VI. Experimental Activities

In this section, we present some of the results we obtained in an extensive experimental activity devoted to validate the approach for AmI planning described in this paper. We recall that we concentrated on the plan generation activity and not on the plan execution activity. For this reason, we use a very simple execution mechanism. The environmental majordomo coordinates the execution of a final plan by requesting the agents to perform the primitive tasks they proposed, according to the defined ordering constraints. Moreover, we do not provide any sophisticate recovery mechanism to tackle situations in which an agent supposed to carry out a primitive task (because the agent proposed it during plan generation) suddenly disconnects during plan execution. In this case, our system simply aborts the execution of the plan and starts a new planning process for the same goal.

As a first experimental situation, we illustrate the performance of the planner in solving the goal `CheckAndRequest (Insulin)` that is intended to check the presence of insulin in the medical store and, if no insulin is left, to make a request to

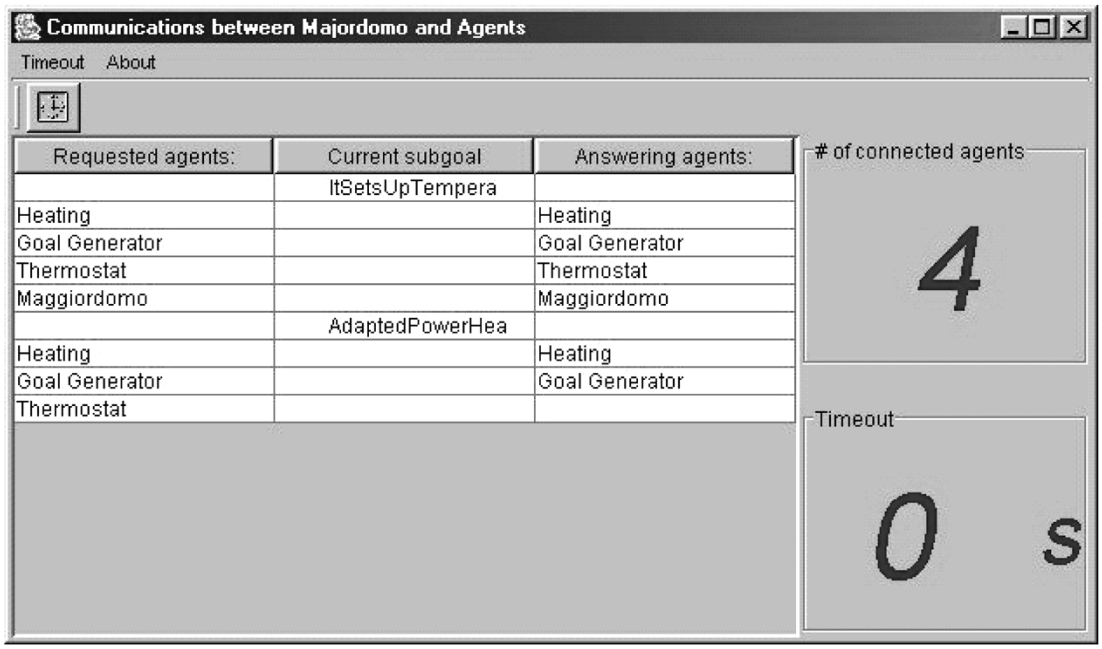


Fig. 7. Monitoring interface of the goal generator agent.

agents	decompositions	indexes
goal generator	$(\texttt{CreateRequestText}(g_1,t_1),[(n_1:\texttt{CreateRequestText}(g_1,t_1))])$	(900,500,500)
	$(\texttt{CreateRequestMessage}(g_1,m_1),[(n_1:\texttt{CreateRequestMessage}(g_1,m_1))])$	(900,500,500)
	$(\texttt{SearchPhoneNumber}(g_1,p_1),[(n_1:\texttt{SearchPhoneNumber}(g_1,p_1))])$	(200,500,500)
	$(\texttt{SearchEmailAddress}(g_1,a_1),[(n_1:\texttt{SearchEmailAddress}(g_1,a_1))])$	(200,500,500)
SMS	$(\texttt{Request}(g_1),[(n_1:\texttt{CreateRequestText}(g_1,t_1)),$ $(n_2:\texttt{SearchCellNumber}(g_1,c_1)),(n_3:\texttt{SendSMS}(t_1,c_1)),$ $(n_1\prec n_3)\wedge(n_2\prec n_3)\wedge(n_1,\texttt{RequestText}(t_1),n_3)\wedge(n_2,\texttt{CellNumber}(c_1),n_3)])$	(900,500,500)
	$(\texttt{SendSMS}(t_1,c_1),[(n_1:\texttt{SendSMS}(t_1,c_1))])$	(900,500,500)
email	$(\texttt{Request}(g_1),[(n_1:\texttt{CreateRequestText}(g_1,t_1)),$ $(n_2:\texttt{SearchEmailAddress}(g_1,a_1)),(n_3:\texttt{SendEmail}(t_1,a_1)),$ $(n_1\prec n_3)\wedge(n_2\prec n_3)\wedge(n_1,\texttt{RequestText}(t_1),n_3)\wedge(n_2,\texttt{EmailAddress}(a_1),n_3)])$	(300,500,500)
	$(\texttt{SendEmail}(t_1,a_1),[(n_1:\texttt{SendEmail}(t_1,a_1))])$	(900,500,500)
fax	$(\texttt{Request}(g_1),[(n_1:\texttt{CreateRequestText}(g_1,t_1)),$ $(n_2:\texttt{SearchPhoneNumber}(g_1,p_1)),(n_3:\texttt{SendFax}(t_1,p_1)),$ $(n_1\prec n_3)\wedge(n_2\prec n_3)\wedge(n_1,\texttt{RequestText}(t_1),n_3)\wedge(n_2,\texttt{PhoneNumber}(p_1),n_3)])$	(200,500,500)
	$(\texttt{SendFax}(t_1,p_1),[(n_1:\texttt{SendFax}(t_1,p_1))])$	(900,500,500)
phone	$(\texttt{Request}(g_1),[(n_1:\texttt{CreateRequestMessage}(g_1,m_1)),$ $(n_2:\texttt{SearchPhoneNumber}(g_1,p_1)),(n_3:\texttt{MakeCall}(m_1,p_1)),$ $(n_1\prec n_3)\wedge(n_2\prec n_3)\wedge(n_1,\texttt{RequestMessage}(m_1),n_3)\wedge(n_2,\texttt{PhoneNumber}(p_1),n_3)])$	(800,500,500)
	$(\texttt{MakeCall}(m_1,p_1),[(n_1:\texttt{MakeCall}(m_1,p_1))])$	(900,500,500)
address book	$(\texttt{SearchPhoneNumber}(g_1,p_1),[(n_1:\texttt{SearchPhoneNumber}(g_1,p_1))])$	(900,500,500)
	$(\texttt{SearchEmailAddress}(g_1,a_1),[(n_1:\texttt{SearchEmailAddress}(g_1,a_1))])$	(900,500,500)
	$(\texttt{SearchCellNumber}(g_1,c_1),[(n_1:\texttt{SearchCellNumber}(g_1,c_1))])$	(900,500,500)
medical store	$(\texttt{IsThere}(g_1),[(n_1:\texttt{IsThere}(g_1))])$	(900,500,500)
environmental majordomo	$(\texttt{CheckAndRequest}(g_1),[(n_1:\texttt{IsThere}(g_1)),(n_2:\texttt{Request}(g_1)),$ $(n_1,\texttt{Present}(g_1))\wedge(\textbf{if }\texttt{Present}(g_1)=\text{false }\textbf{then }n_2)])$	(900,500,500)

Fig. 8. Some of the decompositions initially included in the plan libraries of some of the agents composing the AmI system for our application scenario (the index values refer to performance, cost, and probability of success, respectively).

pharmacies to provide insulin. Note that insulin is a parameter of the `CheckAndRequest` that, therefore, represents a whole set of goals. This goal requires a moderately complex planning process and we use it to give a comprehensive illustration of the behavior of the D-HTN planner. Fig. 8 shows some decompositions initially included by the designer in the plan libraries of some of the agents composing our AmI system. We manually generate the goal (initial task) `CheckAndRequest(Insulin)` by the interface of the goal generator agent (Fig. 9). The environmental majordomo asks the agents currently connected to the AmI system to send their available decompositions. The only decomposition for `CheckAndRequest` is provided by the environmental majordomo itself and introduces two nonprimitive tasks, as shown in Fig. 10.

The two tasks are connected by a selection statement: if the output of the execution of the task `IsThere (Insulin)` (that checks if some insulin is left in the room) is `False` then the task `Request (Insulin)` (that requests to supply insulin) is executed. The planning process picks up the nonprimitive task `IsThere (Insulin)`. This task can be

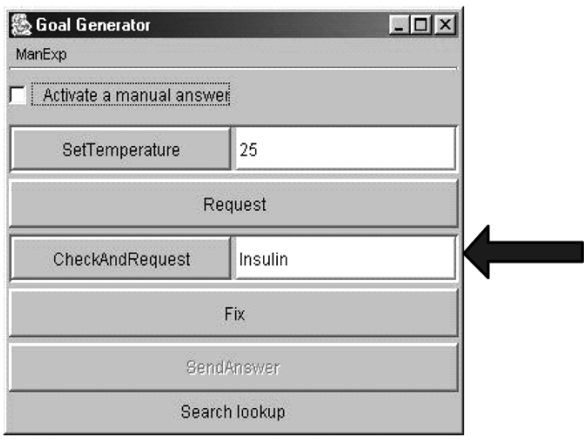


Fig. 9. Generation of the goal `CheckAndRequest (Insulin)`.

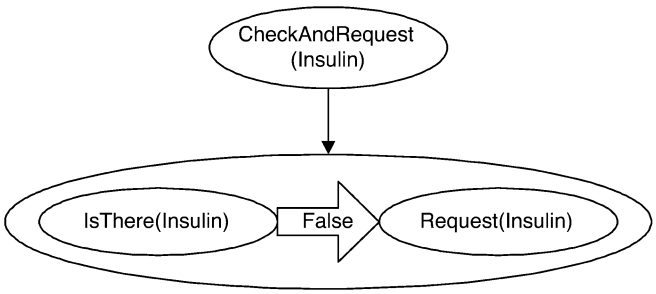


Fig. 10. Task network for the goal `CheckAndRequest (Insulin)` after one planning step.

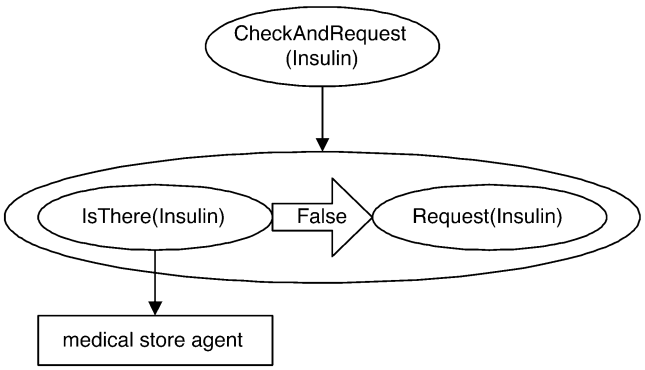


Fig. 11. Task network for the goal `CheckAndRequest (Insulin)` after two planning steps.

decomposed only by a primitive task of the medical store agent (Fig. 11). (Recall that primitive tasks are connected to specific functions of OP semiagents.)

Then, the nonprimitive task `Request (Insulin)` is considered. The SMS, e-mail, fax, and phone agents propose their decompositions for the task in order to contact the pharmacies by different communication means. Since the performance index value of the decomposition proposed by the SMS agent is the highest, this is selected (Fig. 12) because we adopt the heuristic criterion described in Section IV. The selected decomposition constraints the tasks $\texttt{CreateRequestText}(\texttt{Insulin}, t_1)$ and $\texttt{SearchCellNumber}(\texttt{Insulin}, c_1)$ to be executed before $\texttt{SendSMS}(t_1, c_1)$. The first task creates the text t_1 containing the request to supply insulin. The second task finds the mobile phone numbers of the pharmacies. The `SendSMS` task sends out the requests.

The planning process continues and the new inserted nonprimitive tasks are decomposed in primitive tasks performed by the goal generator, the address book agent, and the SMS agent (Fig. 13). The plan is now complete and ready to be executed. The whole planning process takes few seconds on a 1-GHz computer.

The execution of the plan is supervised by the environmental majordomo that requests the agents to perform the primitive tasks they proposed. During the execution, first the medical store is checked for insulin; if it is found its counter is decremented by 1 and the plan execution ends since the task `IsThere (Insulin)` returns `True`; otherwise, the execution continues by activating the other agents. The D-HTN planning process allocates tasks to agents during planning in a way that is more flexible than the static allocation of [45], but more restrictive than the runtime task allocation of [43]. In our system, agents are bound to the nonprimitive tasks they propose during planning, with the advantage that execution of a plan is straightforward. Another possibility could be to bind agents to the actions of a plan at execution time, as in [14]; this solution is more flexible but requires a complex plan executor. We stress that the goal generator agent in our simulated system stands for the monitoring devices on the diabetic patient; in a real system, the operations it performs will be performed directly by these devices. In addition to those presented here, we tested our AmI system in other situations characterizing our application scenario, including keeping the room temperature at a given value (part of this planning activity is shown in Fig. 7), calling a technician to fix the insulin pump, and requesting a telemedicine consult. In all these cases, we obtained results similar to those shown here.

We implemented a very primitive form of learning for improving the ability of the system in solving new problems. At the end of the planning activity, the planner broadcasts to the other agents involved in the plan the decompositions (only those including nonprimitive tasks) that have been employed in the plan. The agents update their plan libraries with the received decompositions. Let us illustrate with an example the utility of updating the plan libraries of the agents. At the end of the construction of the previous plan, the decompositions for `CheckAndRequest` and for `Request` are added to the plan libraries of all the agents involved in the plan, namely to the medical store, goal generator, address book, and SMS agents. In this way, if the medical store agent is part of another AmI system (for example, one oriented to e-commerce), it can exploit the acquired knowledge about `Request` also in the new environment; this is particularly significant if no agent of the new system knows this decomposition. Note that only decompositions including nonprimitive tasks are exchanged at the end of a planning process. Indeed, decompositions including only primitive tasks (being related to the functions of the OP semiagents) are peculiar for each agent. This mechanism for exchanging decompositions is extremely primitive; for example, it is prone to the following problems. When the designer of the system modifies a decomposition m in a decomposition m' by lowering the value of its performance index in the plan library of the environmental majordomo and m has been already

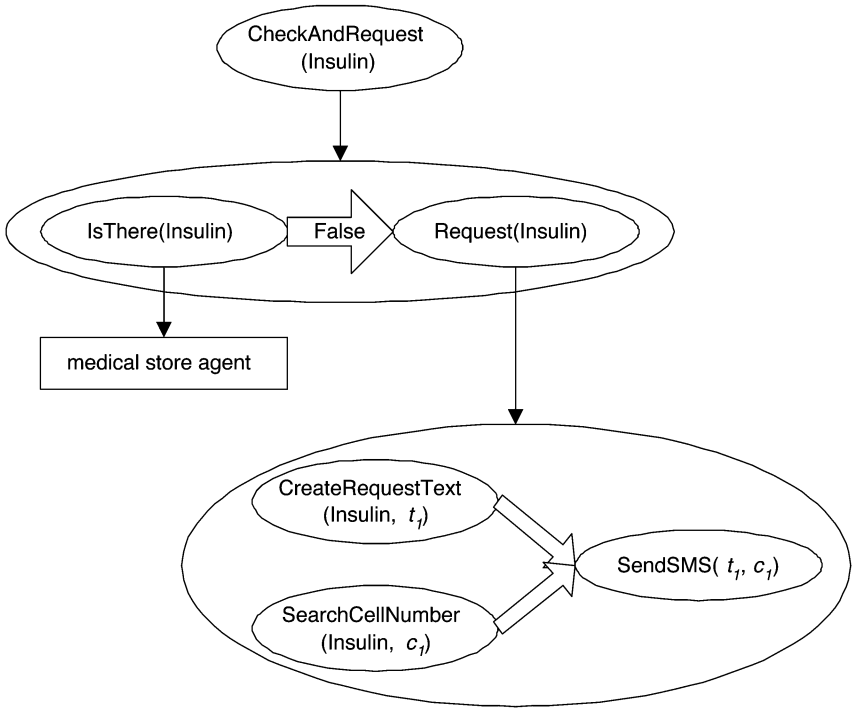


Fig. 12. Task network for the goal `CheckAndRequest (Insulin)` after three planning steps.

copied in the plan libraries of the other agents, m' will never be used in the following plans as long as the agents that could propose m are connected. The same situation appears also when a new agent having very effective (with very high performance value) but incorrect decompositions connects to the system: in this case the decompositions of the new agent will always be employed. The problems outlined above are related to the management of the decompositions and to the uniformity in their definition (especially, of their index values). On the one hand, they show that the current mechanism for knowledge exchanging, although somehow peripheral to the D-HTN planning system we propose in this paper, is too simple and deserves more attention in future work activities. On the other hand, they make it clear that general rules, for example formalized in a specific ontology, for defining the decompositions are needed. The use of ontologies to define concepts of a domain is acquiring paramount importance in multiagent systems and is likely to be a fundamental topic also for AmI applications when interoperability and open systems are concerned. These issues represent very interesting directions for future work.

We now discuss the robustness of our implementation of the D-HTN planner when we manually disconnect an agent during the planning process. We considered the goal `CheckAndRequest` (see above) and we disconnected the SMS agent before it could propose the decomposition for the task `Request`. In this case, the JINI lookup service registers (sometimes with a delay of several seconds) the disconnection and the requests for decompositions performed by the planner reach only the currently connected agents. In this case, the planning process builds a plan similar to that of Fig. 13 in which the decomposition for `Request` proposed by the phone agent substitutes that of the SMS agent. As another example, we disconnected also the address book agent; in this case, the task `SearchPhoneNumber` is decomposed by a primitive task of the goal generator agent that shows an interface to the user who is requested to insert a phone number to call. Of course, when we disconnect the only agent knowing a decomposition that is required by to generate the current plan (for example, the environmental majordomo agent is initially the only one that can provide a decomposition for the `CheckAndRequest` goal in the first example of this section), the planner is no more able to find a plan. The system performs well also when an agent connects to the system during a planning process: in this case, the decompositions of the new agent could be exploited in the next planning phases (note that the portion of the plan already built is not revised to avoid cycles when an agent repeatedly connects and disconnects). These examples show the flexibility of our system in managing situations (typical in AmI applications) characterized by highly dynamic connections and disconnections of the mobile devices. Our implementation of the D-HTN planner is able to build plans that prescribe different courses of actions according to the abilities of the agents currently connected.

In conclusion, the experimental results we obtained demonstrated that the proposed planner and its implementation can effectively address the problems that arise in a realistic AmI application scenario. Moreover, the integration of the middleware services offered by JINI (e.g., the lookup service) with the higher-level planning functions proposed in this paper can be extremely convenient and fruitful to tackle real-world applications.

VII. Conclusion

In this paper, we have addressed the planning problem in AmI applications. The ability to plan a course of actions that devices

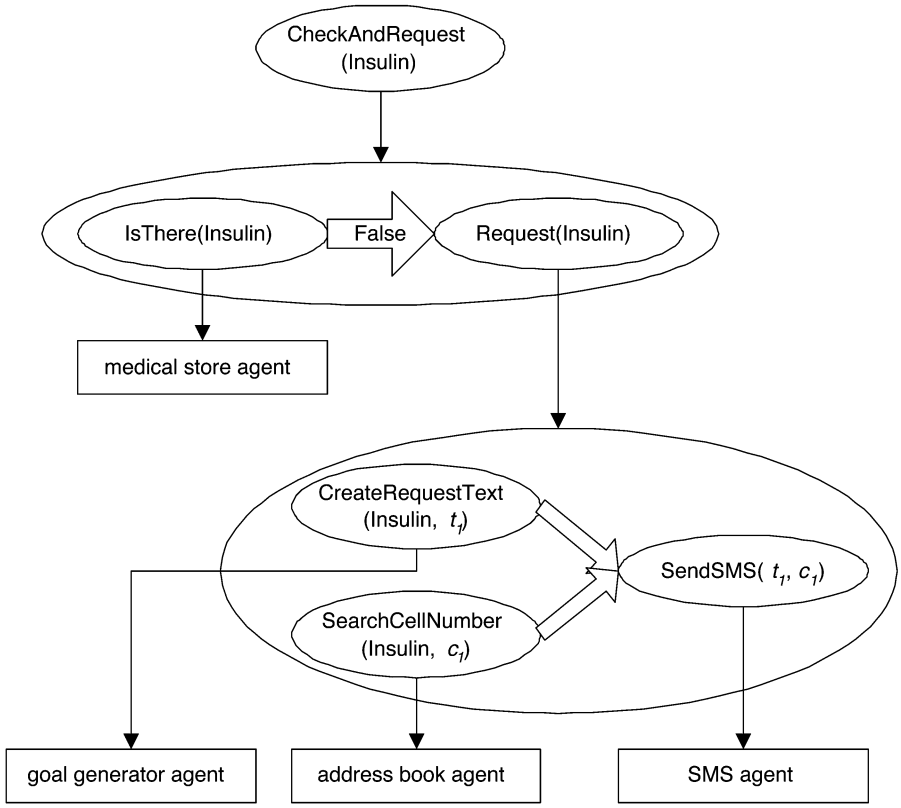


Fig. 13. Complete task network for the goal `CheckAndRequest (Insulin)`.

perform to achieve a given goal appears fundamental for AmI systems that aim at operating in the background of everyday activities to enhance human behavior. We have presented both a critical discussion that justified the development of the D-HTN planner and the issues related to the theory, design, implementation, and experimental validation of the proposed planner embedded in a realistic AmI system. The obtained results demonstrate that D-HTN represents an effective approach to provide AmI systems with goal-oriented capabilities that do not degrade with the dynamic connection and disconnection of the devices, typical of open environments.

The problem of planning in AmI is far from being solved and many of the aspects that we did not address in this paper constitute avenues for future research activities. First of all, as already said, we aim at giving more attention to the execution of a plan and to the coordination of the devices while performing primitive tasks. In particular, the combination of our pure HTN planner with other approaches (such as execution monitoring, replanning, contingency planning, and continuous planning [46, Chapter 12]) could provide solutions for interleaving planning and execution. Moreover, we plan to improve the D-HTN management of the decompositions that are available to the agents; for example to allow their update at runtime, while the system is operating. Along a similar direction, it will be interesting to investigate the learning and knowledge (ontology-based) management techniques that might be employed to improve the D-HTN planner performances. A complete theoretical (similar to the one presented in [20]) and quantitative (including a comparison with other planning approaches) analysis of the D-HTN planner is still missing and requires specific work. Other very important open issues that deserve more attention are related to implementation and concern privacy and security problems (RMI allows only a primitive management of these problems) and the integration of the proposed planner with other middleware solutions for AmI, different from JINI. The difficulties of having a single planner could be overcame by employing a hierarchy of planners to cope with environments that are more complex and structured than a single room. As we have mentioned through this paper, we are also working on an implementation of our AmI system based on real devices for monitoring of patients with cardiac diseases. More generally, future work will be devoted to the improvement of the distributed intelligence techniques to bring the AmI applications more close to the users' exigencies.

ACKNOWLEDGMENT

The authors would like to thank the reviewers for their useful comments, F. Moro and C. Pascali for the fruitful discussions that generated the idea of D-HTN planner, and A. Avigni, M. Bassanelli, M. Corna, G. Grauso, and N. Soresina for their work on the implementation of the system.

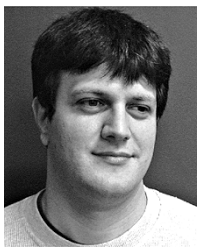
REFERENCES

[1] "Scenarios for Ambient Intelligence in 2010, Final Report," European Commission Information Society Technologies Advisory Group Tech. Rep., 2001.

[2] T. Hasten, L. Benini, A. Chandrakasan, M. Lindwer, J. Liu, R. Min, and F. Zhao, "Scaling into ambient intelligence," in *Proc. IEEE Conf. Design, Automation Test Eur.*, Munich, Germany, Mar. 2003, pp. 76–81.

[3] M. Weiser, "Some computer science issues in ubiquitous computing," *Commun. ACM*, vol. 36, no. 7, pp. 74–84, 1993.

[4] U. Hansrnann, L. Merk, M. J. Nicklous, and T. Stobet, *Pervasive Computing Handbook*. New York: Springer-Verlag, 2001.
[5] R. Berezdivin, R. R. Breinig, and R. Topp, "Next-generation wireless communications concepts and technologies," *IEEE Commun. Mag.*, vol. 40, no. 3, pp. 108–116, Mar. 2002.
[6] J. Yang, W. Holtz, W. Yang, and M. T. Vo, "An adaptive multimodal interface for wireless applications," in *Proc. IEEE Symp. Wearable Comput.*, Santa Cruz, CA, Oct. 1998, pp. 85–90.
[7] S. Kurihara, S. Aoyagi, R. Onai, and T. Sugawara, "Adaptive selection of reactive/deliberate planning for a dynamic environment," *Robot. Auton. Syst.*, vol. 24, pp. 183–195, Sep. 1998.
[8] M. H. Coen, "Design principles for intelligent environments," in *Proc. Nat. Conf. Artif. Intell.*, Madison, WI, 1998, pp. 547–554.
[9] F. Bertamini, R. Brunelli, O. Lanz, A. Roat, A. Santuari, F. Tobia, and Q. Xu, "Olympus: An ambient intelligence architecture on the verge of reality," in *Proc. Conf. Image Anal. Process.*, Mantova, Italy, Sep. 2003, pp. 139–144.
[10] N. Villar, A. Schmidt, G. Kortuem, and H. W. Gellersen, "Interacting with proactive public displays," *Comput. Graph.*, vol. 27, no. 6, pp. 849–857, 2003.
[11] A. Kulkarni, "Design principles of a reactive behavioral system for the intelligent room," *ACM/IEE Bitstream, MIT J. Elect. Eng. Comput. Sci. Student Res.*, pp. 22–26, 2002.
[12] "MIT Artificial Intelligence Laboratory Research Abstracts Tech Rep.," MIT Artificial Intelligence Lab., Cambridge, MA, Sep. 2002.
[13] G. Look, J. Sousa, and M. Kim, "Planlet: Supporting Plan Based User Assistance," Intel Research, Tech. Rep. IRS-TR-03-005, Apr. 2003.
[14] G. Look and S. Peters, "Plan-driven ubiquitous computing," in *Student Oxygen Workshop*. Cambridge, MA, 2003.
[15] Z. J. Haas, "A communication infrastructure for smart environments: A position article," *IEEE Pers. Commun.*, vol. 7, no. 5, pp. 54–58, Oct. 2000.
[16] E. H. Durfee, "Distributed problem solving and planning," in *Multiagent Systems: A Modem Approach to Distributed Artificial Intelligence*, G. Weiss, Ed. Cambridge, MA: MIT Press, 1999, ch. 3, pp. 121–164.
[17] B. Schilit, N. Adams, and R. Want, "Context-aware computing applications," in *Proc. IEEE Workshop Mobile Comput. Syst. Applicat.*, Santa Cruz, CA, Dec. 1994, pp. 85–90.
[18] N. Hanssens, A. Kulkarni, R. Tuchinda, and T. Horton, "Building agent-based intelligent workspaces," in *ABA Conf.*, Las Vegas, NV, Jun. 24–27, 2002.
[19] F. Gouaux *et al.*, "Ambient intelligence and pervasive systems for the monitoring of citizens at cardiac risk: New solutions from the epi-medics project," in *Proc. IEEE Conf. Comput. Cardiol.*, Memphis, TN, Sep. 2002, pp. 289–292.
[20] K. Erol, J. Hendler, and D. S. Nau, "Complexity results for HTN planning," *Ann. Math. Artif. Intell*, vol. 18, no. 1, pp. 69–93, 1996.
[21] G. Weiss, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA: The MIT Press, 1999.
[22] M. Wooldridge, *An Introduction to Multiagent Systems*. Chichester, UK: Wiley, 2002.
[23] F. Ramparanay, O. Boissier, and H. Brouchoud, "Cooperating autonomous smart devices," presented at the Smart Object Conf., Grenoble, France, May 2003.
[24] P. Busetta, M. Merzi, S. Rossi, and F. Legras, "Real-time coordination for ambient intelligence," presented at the Workshop Represent. Approaches Time-Critical Decentralized Resource/Role/Task, Melbourne, Australia, July 2003.
[25] G. Evans, "Solving home automation problems using artificial intelligence techniques," in *Proc. IEEE Conf. Consumer Electron.*, Rose-mont, IL, June 1991, pp. 140–141.
[26] J. A. Weaver, "Artificial intelligence in the domestic environment," in *Proc. IEE Colloquium Intell. Consumer Products*, London, UK, Dec. 1989, pp. 1/1–1/3.
[27] JINI (2004). [Online]. Available: http://wwws.sun.com/software/jini/
[28] E. Braunwald, A. S. Fauci, D. L. Kasper, S. L. Hauser, D. L. Longo, and J. L. Jameson, *Harrison's Principles of Internal Medicine*. New York: McGraw-Hill, 2001.
[29] H. H. Asada, P. Shaltis, A. Reisner, S. Ree, and R. C. Hutchinson, "Mobile monitoring with wearable photopletismographic biosensors," *IEEE Eng. Med. Biol. Mag.*, vol. 22, no. 3, pp. 28–40, May-June 2003.
[30] F. Amigoni, M. Dini, N. Gatti, and M. Somalvico, "Anthropic agency: A multiagent system for physiological processes," *Artif. Intell. Med.*, vol. 27, no. 3, pp. 305–334, Mar. 2003.
[31] I. Korhonen, J. Parkka, and M. V. Gils, "Health monitoring in the home of the future," *IEEE Eng. Med. Biol. Mag.*, vol. 22, no. 3, pp. 66–73, May/June 2003.
[32] S. J. Alcock, "Technology for continuous invasive monitoring of glucose," in *Proc. IEEE Conf. Eng. Med. Biol.*, Amsterdam, The Netherlands, Oct.–Nov. 1996, pp. 2156–2158.
[33] D. S. Weld, "An introduction to least commitment planning," *AI Mag.*, vol. 15, no. 4, pp. 27–61, Winter 1994.
[34] M. E. des Jardines, E. H. Durfee, C. L. Hortiz, and M. J. Woloverton, "A survey of research in distributed, continual planning," *AI Mag.*, vol. 20, pp. 13–22, 1999.
[35] D. S. Weld, "Recent advances in AI planning," *AI Mag.*, vol. 20, no. 2, pp. 93–122, 1999.
[36] A. Tate, "Using goal structure to direct search in a problem solver," Ph.D. dissertation, Univ. Edinburgh, Edinburgh, UK, 1975.
[37] E. Sacerdoti, *A Structure for Plans and Behavior*. New York City: North Holland, Elsevier, 1977.
[38] K. Konolige, "A first-order formalization of knowledge and action for a multiagent planning system," in *Machine Intelligence 10*, J. E. Hayes, D. Michie, and Y. H. Pao, Eds. Chichester, UK: Ellis Horwood, 1982.
[39] R. G. Smith, "The contract net protocol: High-level communication and control in a distributed problem solver," *IEEE Trans. Comput.*, vol. C-29, pp. 1104–1113, Dec. 1980.
[40] P. Cohen and H. Levesque, "Intention is choice with commitment," *Artif. Intell.*, vol. 42, no. 3, pp. 213–261, 1990.
[41] S. Kraus, E. Ephrati, and D. Lehmann, "Negotiation in a noncooperative environment," *J. Exp. Theor. Artif. Intell.*, vol. 3, no. 4, pp. 255–282, 1991.
[42] P. Stone, *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer*. Cambridge, MA: MIT Press, 2000.
[43] E. Durfee and V. Lesser, "Partial global planning: A coordination framework for distributed hypotesis formation," *IEEE Trans. Syst., Man, Cybern.*, vol. 21, pp. 1167–1183, Sep.–Oct. 1991.
[44] M. E. Pollack, "Planning in dynamic environments: The DIPART system," in *Proc. Adv. Planning Technol.: Technol. Achievements ARPA/Rome Laboratory Planning Initiative*. Menlo Park, CA, 1996, pp. 218–225.
[45] D. Corkill, "Hierarchical planning in a distributed environment," in *Proc. Int. Joint Conf. Artif. Intell.*, Tokyo, Japan, Aug. 1979, pp. 169–175.
[46] S. Russell and P. Norvig, *Artificial Intelligence: A Modem Approach*. Englewood Cliffs, CA: Prentice-Hall, 2003.
[47] F. Moro, C. Pascali, C. Pinciroli, and M. Roveri, "A Hybrid Planner for an Environmental Agency Tech. Rep. 2003–41," Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milan, Italy, 2003.
[48] F. Amigoni and M. Somalvico, "Dynamic agencies and multi-robot systems," in *Distributed Autonomous Robotic Systems 3*, T. Lueth, R. Dillmann, P. Dario, and H. Worn, Eds. Munich, Germany: Springer-Verlag, 1998, pp. 215–224.
[49] F. Amigoni and N. Gatti, "An environmental multiagent architecture for health management," in *Proc. Ital. Assoc. Artif. Intell. Workshop Ambient Intell.*, Pisa, Italy, Sep. 2003, pp. 58–69.
[50] Sun Microsystems. (2004) Remote Method Invocation. [Online]. Available: http://java.sun.com/products/jdk/rmi/
[51] Echelon. (2004) Lonworks. [Online]. Available: http://www.echelon.com/products/lonworks/default.htm
[52] G. P. Picco and A. Murphy, "Reliable communication for highly mobile agents," *Auton. Agent. Multi Agent. Syst.*, vol. 5, no. 1, pp. 81–100, 2002.



Francesco Amigoni (S'98–A'00) was born in Soresina (CR), Italy, on December 2, 1971. He received the Laurea degree in computer engineering and the Ph.D. degree in computer engineering and automatica from the Politecnico di Milano, Milan, Italy, in 1996 and 2000, respectively.

From December 1999 to September 2000, he was a Visiting Scholar in the Computer Science Department, Stanford University, Stanford, CA. Since February 2002, he has been an Assistant Professor in the Dipartimento di Elettronica e Informazione, Politecnico di Milano. He is the author of about 30 papers published in international journals, books, and conference proceedings. His main research interests include agents and multiagent systems, mobile robotics, and the philosophical aspects of artificial intelligence.

Nicola Gatti (M'03) was born in Milano, Italy, in 1976. He received the Laurea degree in biomedical engineering in 2001 from the Politecnico di Milano, Milan, Italy, where he is currently pursuing the Ph.D. degree in information engineering.

He is a member of the Artificial Intelligence and Robotics Laboratory, Politecnico di Milano. His main research interests include agents and multiagent systems, cooperative negotiation, and robotics for disabled people.

Carlo Pinciroli was born in Varese, Italy, in 1980. He is currently pursuing the Laurea degree in computer science at the Politecnico di Milano, Milan, Italy, and the M.S. degree in computer science at University of Illinois, Chicago.

He is a member of the Artificial Intelligence and Robotics Laboratory in Politecnico di Milano and cooperates with Cefriel, Milan, Italy. His main interests include agent and multiagent systems, philosophical aspects of artificial intelligence, machine learning and machine vision systems, computer architecture, and embedded systems.

Manuel Roveri received the Dr. Ing. degree in computer engineering in 2003 from the Politecnico di Milano, Milan, Italy, where he is currently pursuing the Ph.D. degree in computer engineering, and the M.S. degree in computer science from the University of Illinois, Chicago, in 2003.

His main research interests include adaptive and evolutionary technologies and methodologies for analysis, processing, and classification of images by computational intelligence approaches.