Designing Autonomous Robots using GOLEM

Fabrizio Messina, Giuseppe Pappalardo, Corrado Santoro
University of Catania – Dept. of Mathematics and Computer Science
Viale Andrea Doria, 6 — 95125 - Catania, ITALY
EMail: {messina,pappalardo,santoro}@dmi.unict.it

***Abstract*—In this paper we present a goal model for the design of autonomous robots. A framework is proposed which allows a developer to design the behaviour of a robot in accordance with a *rational model*, which is conceived to let the autonomous system to mimic the human behaviour. The framework, called GOLEM, is based on the abstractions of *goals* and *sub-goals*, which are combined through specific *relationships* to specify the robot's activities; such activities, assembled together, form the rational behaviour of the autonomous system. The central motivation for the design of our framework is represented by the central issue of integrating full *awareness* and *autonomy*. Indeed, in the proposed solution the execution of goals does not obey a pre-fixed sequence; rather, the next goal to achieve is selected, each time, on the basis of an evaluation of its *opportunity*.**

I. Introduction

Programming the behaviour of autonomous robots requires powerful abstractions, languages and suitable frameworks by which the designer can implement behavioural elements featuring *autonomy* and *situatedness*. By means of suitable specific constructs, expressing the actions to be undertaken by the robot to reach the goal, for which the robot itself has been designed, can be easier.

Autonomy is a special, fundamental aspect to be taken into account, as it is the *ability of a machine to select the right action(s) to be performed to reach a certain goal* without relying to any additional (human) control. Selecting the appropriate actions is an activity to be performed taking into account the reference environment which, being in many cases the physical and real world, is *dynamic* and often *unpredictable*. For the reasons above (dynamic and unpredictability of the environment) an action *may fail*. Indeed, since the environment is dynamic, its state, which has been perceived before starting the action, is changed, thus making the action no more feasible. In this case, as humans being, a real autonomous robot should be able to identify such a situation and adopt countermeasures, if any.

In order to deal with the issue discussed so far, the software implementing the robot's behaviour must be properly designed. In general, the underlying model of a robot behaviour is modelled as a continuous loop, involving the three phases *sense*/*evaluate*/*act*: here, the action to be performed is selected on the basis of the state of the environment (*sense* phase) and the state of the robot itself[1] (*evaluate* phase). At first sight such a model could be easily implemented with a loop calling functions for environment sensing and using a series of "*if*"s to let the program select the right action(s). Another programming model which is often used in the field of autonomous systems considers the use of *finite-state machines* [2], [1] or *state-charts* [6], [12], which help the modelling of a robot's behaviour by exploiting a graphical approach which can then be implemented either directly or using proper software libraries [10], [3]. Other proposals exploit the concept of a *reactive rule* [4], [11], [5], thus providing languages or platforms to express robot behaviour by means of a series of rules in the form *event*/*condition*/*action*.

[1] In rational robots, this state can include also a form of "knowledge".

Many of the solutions described above are derived from *computational models*, which aim at trying to map computer science abstractions to a framework which lets a robot exhibit an autonomous behaviour. Nevertheless we followed a different approach, consisting in starting from a *rational model*, i.e. to understand how a human would behave in order to autonomously reach a goal, in order to derive a proper computational model capable of—more or less precisely—represent it. As a consequence, in this work we introduces an abstract framework, called GOLEM, in which a robot's behaviour is decomposed into a set of *tasks* and *sub-tasks* (which are indeed referred to as *goals*), properly interconnected to one another on the basis of certain *dependency* relationship. The order of execution of tasks is not a-priori fixed, but the task to be executed is dynamically chosen by means of a *dynamic scheduling policy*, which takes into account the concepts of *feasibility*, *priority* and *opportunity*. The idea is to endow the robot with a high degree of *deliberation ability*, by letting it autonomously decide which (sub-)task or (sub-)goal adopt at each time instant.

The paper is structured as follows. Section II provides a real-life scenario which is used as a motivating example to introduce the model. Section III formalises the GOLEM framework, illustrating the entities, the relationships and the execution semantics. Section IV describes the GOLEM version of the scenario introduced in Section II. Section V deals with related work. Section VI ends up the paper with our conclusions.

II. A daily life example

Supposing we should program a robot to perform a typical daily task performed by humans: we wish the robot behave in the *same way* as a human would do, so as to make the robot exhibit a human-like (and in some sense "rational") behaviour. Starting from such an example is very useful to derive some interesting properties which eventually are the basis of the framework described in the subsequent Sections of the paper.

A. The "Food Buyer" Robot

Let us suppose we want to program a humanoid robot to go to the supermarket and buy some food for us, given a specific food list. In detail, we need some bread, milk, beer and pasta;

moreover, for the beer, we would like to specify an order of preference for the brand, i.e. first *Brand1*, if available, then *Brand2* as a second choice, etc.

We start by specifying the *overall goal*, say GR, as **to buy the food in the list**. Moreover, it is natural to express GR by splitting it in a set of *sub-goals* to be achieved by the robot, as in the following sequence:

$G1$	First, go to the supermarket.
$G2$	Pick each food type specified in the list, and put it into the cart.
$G3$	When each item of the list has been picked, go to the checkout and wait in the line.
$G4$	When there are no more people before you, pay for the food which is the cart.
$G5$	Finally, go back home and bring me the food.

We can say that fulfilling sub-goals $\{G1 \dots G5\}$ implies the automatic achievement of the overall goal GR, which is a way of reasoning very close to that of humans.

$G2$ can be decomposed into a set of sub-goals: $G21$ for **bread**, $G22$ for **milk**, $G23$ for **beer** (with brand preference), and $G24$ for **pasta**. As a consequence, the robot has to find and reach the proper stand[2] for each item specified in the list above in order to pick the item. Just as a human might reason, the order in which sub-goals $\{G21 \dots G24\}$ are fulfilled does not matter: it is up to the robot (as for a human) to make such a choice: it only matters that all items, sooner or later, will be picked. More in detail, a person would make a *rational choice* on the basis, for example, of the distance among the various stands, thus trying to minimise the length of the path, i.e. it could apply a sort of policy, or a strategy.

Moreover, irrespective of the specific policy used to perform the choice, it is important to remark the difference between the set of sub-goals $\{G1 \dots G5\}$ and $\{G21 \dots G24\}$: in the former case, the order of execution **matters** (strict sequence, no autonomous choice is possible); in the latter case, the robot can autonomously decide the order following any—more or less rational—personal criteria. In making such a choice, different aspects could be considered, some of them subjective, and others, undoubtedly, objective; one of the latter is obviously the feasibility of the sub-goal: e.g., if, at a certain time instant, the stand of the beers is not reachable due to too much confusion, sub-goal $G23$ is not feasible and thus it is useless to choose it. Such an infeasibility is indeed temporary, for we could try to achieve $G23$ after some time, as soon as, e.g., there is no more crowd.

Order of execution is not the sole difference between sets $\{G1 \dots G5\}$ and $\{G21 \dots G24\}$; while, as stated above, achievement of GR requires the *mandatory* achievement of *all* of its sub-goals $\{G1 \dots G5\}$, the same cannot be said for $G2$ and $\{G21 \dots G24\}$: indeed, if a certain food is unavailable (e.g., milk), the robot should nevertheless proceed with the shopping, that is, to consider $G2$ achieved the successful completion of only some it sub-goals suffices.

Also goal $G23$ deserves a special remark; here the objective is to pick some beer based on brand preferences. To this aim, we can model $G23$ as further composed of a set of sub-goals, one for each specific brand, e.g. $GB1$ for **Brand 1**, $GB2$ for **Brand 2**, $GB3$ for **Brand 3**.

The sense of this (de)composition, in terms of selection and achievement, is different from both GR and $G2$: if $GB1$ succeeds (Brand 1 is available), then $G23$ is achieved; otherwise try $GB2$ and so on.

B. Discussion

The above example shows some interesting properties related to the "rational behaviour" of an autonomous entity (being either a human or a robot).

First of all, the behaviour can be represented as a set of *goals*, some of which, in turn, can be further decomposed into *sub-goals*. Making such a decomposition allows the identification of the various tasks and actions to be accomplished by the robot, together with their *relationships*.

The *relationship* plays a fundamental role in establishing the *order of execution* and the *achievement policy*. Indeed, order of execution can be strictly in sequence in some cases, but, in many other cases, which goal to try and achieve is (or can be) the robot's autonomous choice, which can be performed on the basis of some parameters like (for example):

- the state of the environment or the robot itself ("let's go and get pasta first, because it is nearer to my position");
- past experience of the robot ("let's get pasta because I know where it is located");
- rational opportunities ("let's put the beer last in the cart, to lower the probability of breaking the bottles");
- other reasons.

A goal (or sub-goal) can *succeed* or *fail*. Failure can be due to various reasons, whether *temporary*, e.g. crowd in a certain area of the supermarket, or *permanent*, e.g. there are no items of a certain food type. A goal failed for a temporary reason can be retried later, when the robot "thinks" there is again an opportunity to execute it.

All of these aspects contribute to conferring the robot a complete awareness of its objectives, as well as the ability to *autonomously deliberate* about the actions to be performed at each time instant. All of these aspects are the basis of the GOLEM framework which is described in the next Section.

III. The GOLEM framework

GOLEM, which is the framework presented in this paper, consists of a set of precise rules specifying the syntax to which a robot program P must obey to be correctly executed, and an *abstract machine* (GAM, GOLEM Abstract Machine) which, in turn, has the responsibility of executing P, basing on some semantics rules.

[2]We are supposing that the robot has a knowledge of the physical location of goods in the supermarket.

$$
\begin{array}{rcll}
P & ::= & g & \\
g & ::= & sg \mid cg & \\
sg & ::= & (f, p, act) & \\
f & ::= & functor \rightarrow \{true, false\} & \\
p & ::= & functor \rightarrow \mathcal{N} & \\
act & ::= & functor \rightarrow status & \\
status & ::= & ACHIEVED & \\
 & & \mid T\_FAIL & \\
 & & \mid P\_FAIL & \\
cg & ::= & (rel, g\_set) & \\
rel & ::= & ALL & \\
 & & \mid ALL\_SEQ & \\
 & & \mid AT\_LEAST(k) & \\
 & & \mid SEQ\_UNTIL & (k \geq 1) \\
g\_set & ::= & \{g_1, \ldots, g_n\} & (n > 1)
\end{array}
$$

Fig. 1. Basic GOLEM Syntax

A. GOLEM Program

As shown in Figure 1, which reports the basic GOLEM syntax, a robot program P is represented by its *main goal*, g. A goal g, in turn, represents the *tasks that the robot must do to achieve a certain state*. Moreover, according to the nature of such tasks, a goal can be *simple* or *composite*.

A *simple goal*, sg, represents a goal which requires no specific further decision on the actions to be undertaken. A simple goal includes a computation which is executed "as is" to achieve the goal itself; if the computation ends with success, the goal is achieved, otherwise, it is failed. A simple goal is represented with a tuple composed of the following parts:

- *feasibility function*, f;
- *opportunity evaluator*, p;
- *goal action*, act;

Feasibility function contains the necessary logic to evaluate whether the goal is feasible or not. The internal structure of f is not specified, it is supposed to be a program code which senses the state of the environment and/or robot, performs suitable evaluations and returns a boolean value indicating the feasibility of the goal. The idea behind the concept of feasibility is to verify that there are no conditions that *would impede* the achievement of a goal; indeed, stating that a goal is feasible does not imply that the goal will surely succeed: other situations might happen, during the execution of the tasks of the goal, which cause the failure of a specific action. For example, feasibility of goal $G4$ in the supermarket example (checking out) depends on an empty checkout line.

Opportunity Evaluator is a function by which the designer is able to specify the level of priority or importance of the goal, with respect to the others. Such importance can be represented by a number, e.g. an integer. As a consequence this value is used to select, among different feasible goals, the one to execute. Such a measure is not intended to be a static value, as it should be dynamically evaluated before deciding which goal to execute. Evaluating the opportunity could thus imply to sense the state of the environment (or the robot itself) and, on this basis, decide if the goal is, in this time instant, more important than another one (or the most important one). As an example, in order to try to minimise the path in our supermarket example, the opportunity function of goals $G21$, $G22$, $G23$ and $G24$ could be a factor proportional to the distance to the (respective) stand to be reached. To perform a correct evaluation, a proper *metric* has to be derived to map the (notion of) importance of a goal to a positive integer: the higher the value the higher the importance of the goal.

Goal action is an entity representing the actions to be performed in order to try to achieve the goal itself. It is a piece of code including the required statements to make the robot carry out the intended actions. From a conceptual point of view, this code should not present rational choices or deliberative actions, which instead should be modelled as sub-goals of a composite goal (see below). Again, goal action is modelled as a function returning three possible constants:

- **ACHIEVED**. Execution of the action caused the successful achievement of the goal.
- **T_FAIL**. Execution failed, but such a failure is considered *temporary*, i.e. retrying the execution later could lead to success.
- **P_FAIL**. Execution failed and such a failure is considered *permanent*.

A composite goal, cg, is instead represented as a pair (rel, g_set) comprising a *set of sub-goals* g_set and a *relationship condition* rel, which can be one of the following:

- **ALL**. The goal succeeds when *all* of its sub-goals are achieved; the order in which the sub-goals are achieved does not matter (i.e. the set g_set is *not ordered*).
- **ALL_SEQ**. The goal succeeds when *all* of its sub-goals are achieved but the order in which the sub-goals are achieved must be a strict sequence (i.e. the set g_set is *ordered*).
- **AT_LEAST(k)**. The goal succeeds when *at least* k of its sub-goals (with k specified) are achieved; the order of achievement does not matter.
- **SEQ_UNTIL**. The goal succeeds when (as soon as) *any sub-goal* is achieved; the set g_set is *ordered* and sub-goal achievement is tried in strict sequence.

Each sub-goal may be, in turn, either simple or composite. The result is thus a hierarchy of goals, with proper relationships, which represents the model of the overall autonomous behaviour of the robot. Such a hierarchy is handled by the GOLEM Abstract Machine in order to execute the goals respecting the semantics of relationships, as it is described in the next Subsection.

B. GOLEM Abstract Machine

A GOLEM robot program P is executed by the *GOLEM Abstract Machine* (GAM) whose behaviour is essentially based on a continuous loop iterating on the following steps:

1) Evaluation of feasible goals.
2) Evaluation of their opportunity values.
3) Selection of the feasible goal with highest opportunity value.

4) Execution of the selected goal.

The GAM stops executing the program when P either has been *successfully achieved* or has *permanently failed*.

For each iteration, the program P is scanned by looking into its composite goals, also descending into the hierarchy if needed, with the objective of finding a simple (sub-)goal which can be executed; this is performed using proper policies which depend on the relationship type of composite goals. At each iteration, when execution of the goal is completed, it is checked whether goal execution caused the achievement of P or its permanent failure. In the former case, the program ends with success, otherwise an error is reported.

C. GAM state information and main functions

The GAM has an internal state represented by the tuple (P, AG, FG, TFG) where P is the robot program, AG is the set of *achieved goals*, FG is the set of *permanently failed goals*, and TFG is the set of *temporarily failed goals*.

Sets AG, FG and TFG are initially empty and then properly modified during program execution; they may contain only simple goals, while evaluation of achievement or failure of a composite goal is performed by analysing its composing sub-goals. To this aim, the GAM exploits two functions: *IS_ACHIEVED*$(g) \rightarrow \{true, false\}$ and *IS_P_FAILED*$(g) \rightarrow \{true, false\}$. The former evaluates whether a goal g has been successfully achieved or not; Table I reports the specific behaviour of this function which depends on the goal type (simple or composite) and the relationship type (for composite goals). The latter evaluates whether a goal g has to be considered permanently failed; Table II reports the behaviour of such a function. In all tables, for convenience, we assume that, when g is composite, $g.g\_set$ represents its goal set. Likewise, if g is simple, $g.p$, $g.f$ and $g.act$ represent g's feasibility function, opportunity evaluator and goal action, respectively.

$IS\_ACHIEVED(g) \rightarrow \{true, false\}$

Goal Type/Rel.	Function Behaviour
$simple$	$g \in AG$
ALL	$\bigwedge_{h \in g.g\_set}\{IS\_ACHIEVED(h)\}$
ALL_SEQ	$\bigwedge_{h \in g.g\_set}\{IS\_ACHIEVED(h)\}$
$AT\_LEAST(k)$	$\|\{h \in g.g\_set : IS\_ACHIEVED(h)\}\| \geq k$
SEQ_UNTIL	$\bigvee_{h \in g.g\_set}\{IS\_ACHIEVED(h)\}$

TABLE I. *IS_ACHIEVED* FUNCTION

$IS\_P\_FAILED(g) \rightarrow \{true, false\}$

Goal Type/Rel.	Function Behaviour
$simple$	$g \in FG$
ALL	$\bigvee_{h \in g.g\_set}\{IS\_P\_FAILED(h)\}$
ALL_SEQ	$\bigvee_{h \in g.g\_set}\{IS\_P\_FAILED(h)\}$
$AT\_LEAST(k)$	$\|\{h \in g.g\_set : IS\_P\_FAILED(h)\}\| > \|g.g\_set\| - k$
SEQ_UNTIL	$\bigwedge_{h \in g.g\_set}\{IS\_P\_FAILED(h)\}$

TABLE II. *IS_P_FAILED* FUNCTION

Steps 1 and 2 of the GAM execution loop perform evaluation of goals in order to select a feasible one. We assume a function $EVAL(g)$ that evaluates both the feasibility and opportunity of a goal g as follows. If g is not feasible, or belongs to either AG or FG, the $EVAL(g)$ returns $-\infty$; otherwise it returns the value of the opportunity function $g.p()$ if g is simple, or a proper evaluation of its sub-goals if g is composite. Its specific behaviour is reported in Table III.

$EVAL(g)$

Goal Type/Rel.	Function Behaviour
any	$-\infty$ **if** $g \in AG \cup FG$
$simple$	**if** $g.f()$ **then** $-\infty$ **else** $g.p()$
ALL	$max\{EVAL(h) \| h \in g.g\_set\}$
ALL_SEQ	$EVAL(first\_of(\{h\| \in g.g\_set - (AG \cup FG)\}))$
$AT\_LEAST(n)$	$max\{EVAL(h) \| h \in g.g\_set\}$
$SEQ\_UNTIL(n)$	$EVAL(first\_of(\{h\| \in g.g\_set - (AG \cup FG)\}))$

TABLE III. *EVAL* FUNCTION

D. GAM algorithm

The functions introduced above are exploited by the GAM to execute a program P. Algorithm 1 reports the pseudo code of the main GAM behaviour: here a loop is executed until the program P is considered achieved or permanently failed; the body of the loop evaluates the feasibility of P and, if this is the case, it executes P by calling EXEC_FEASIBLE.

Algorithm 1 GAM Main Algorithm

```
procedure GAM(P:goal)
    $AG \leftarrow \emptyset$; $FG \leftarrow \emptyset$; $TFG \leftarrow \emptyset$
    while ¬IS_ACHIEVED(P)∧¬IS_P_FAILED(P) do
        if EVAL(P) $\neq -\infty$ then
            EXEC_FEASIBLE(P)
        end if
    end while
end procedure
```

Procedure EXEC_FEASIBLE, whose pseudo-code is illustrated in Algorithm 2, checks the goal type, i.e. whether it is simple or composite: in the former case, the $act()$ function is called and its outcome is tested in order to update sets AG, FG and TFG; in the latter case, a proper procedure is called to perform execution according to the relationship type, as detailed in the following:

1) ALL relationship: The ALL relationship requires that *all* subgoals must be executed but the order of execution does not matter. The EXEC_ALL procedure, reported in Algorithm 3, determines the set of feasible sub-goals and, from it, selects the one with the highest opportunity value; if this condition holds for more than a goal, a non-deterministic (random) choice is performed (this is done by the ONE_OF function in line 11). Once the sub-goal has been selected, the EXEC_FEASIBLE procedure is called again to concretely execute the goal.

2) ALL_SEQ relationship: This kind of relationship is more strict than the previous one: not only all sub-goals must be executed, but also the order of execution must be respected. As it is illustrated in Algorithm 4, the procedure EXEC_ALL_SEQ scans the set of sub-goals in sequence in order to find the first non-achieved sub-goal: if it is feasible, the sub-goal is directly executed.

3) AT_LEAST(k) relationship: This relationship is equivalent to ALL, with the sole exception that the composite goal is considered achieved when at least k sub-goals succeed. This condition is stated in the *IS_ACHIEVED* function, while, with respect to sub-goal selection, the policy here is to find

Algorithm 2 Execute a feasible goal

```
procedure EXEC_FEASIBLE(g:goal)
                              ▷ This procedure supposes that P is feasible
    if IS_SIMPLE(g) then
        $r \leftarrow g.act()$
        if $r = ACHIEVED$ then
            $AG \leftarrow AG \cup \{g\}$;   $TFG \leftarrow TFG \setminus \{g\}$
        else if $r = P\_FAILED$ then
            $FG \leftarrow FG \cup \{g\}$;   $TFG \leftarrow TFG \setminus \{g\}$
        else                                          ▷ $T\_FAILED$
            $TFG \leftarrow TFG \cup \{g\}$
        end if
    else
        $(rel, g\_set) \leftarrow g$
        if $rel = ALL$ then
            EXEC_ALL($g\_set$)
        else if $rel = ALL\_SEQ$ then
            EXEC_ALL_SEQ($g\_set$)
        else if $rel = AT\_LEAST(k)$ then
            EXEC_ALL($g\_set$)
        else if $rel = SEQ\_UNTIL$ then
            EXEC_SEQ_UNTIL($g\_set$)
        end if
    end if
end procedure
```

Algorithm 3

```
procedure EXEC_ALL(g_set:set of goal)
    $failed \leftarrow \{g \in g\_set : IS\_P\_FAILED(g)\}$
    if $failed \neq \emptyset$ then
        return
    end if
    $selectables \leftarrow \{g \in g\_set : EVAL(g) \neq -\infty\}$
    if $selectables = \emptyset$ then
        return
    end if
    $candidates \leftarrow \{g \in selectables : max\ EVAL(g)\}$
    $selected \leftarrow$ONE_OF($candidates$)
    EXEC_FEASIBLE($selected$)
end procedure
```

Algorithm 4

```
procedure EXEC_ALL_SEQ($g\_set : set\ of\ goal$)
    for $g \in g\_set$ do
        if $g \in FG$ then
            return
        end if
        if $g \notin AG$ then
            if $EVAL(g) \neq \infty$ then
                EXEC_FEASIBLE($g$)
            end if
            return
        end if
    end for
end procedure
```

the feasible sub-goal with the highest opportunity value and execute it. Such a behaviour is thus the same of the EXEC_ALL procedure, which is the one called also in this case as it is shown in Algorithm 2.

4) SEQ_UNTIL relationship: Also in this case, the set of sub-goals is ordered thus the policy adopted for execution is to scan such a set in order to find the first feasible goal, also ensuring that all the previous ones have already failed. Such a behaviour is reported in the EXEC_SEQ_UNTIL procedure illustrated in Algorithm 5.

Algorithm 5

```
procedure EXEC_SEQ_UNTIL(g_set:set of goal)
    for $g \in g\_set$ do
        if $g \in AG$ then
            return
        end if
        if $g \notin FG$ then
            if $EVAL(g) \neq \infty$ then
                EXEC_FEASIBLE($g$)
            end if
            return
        end if
    end for
end procedure
```

IV. Case-Study

In order to show a practical use of the GOLEM framework, as well as the related benefits, let us provide a case-study based on the example provided in Section II. We report the complete set of goals in the tree shown into Figure 2, in which some nodes represent the (sub)goals, while some others represent the relationship by which the (sub)goal must be executed. In order to discuss the details of the case study, in the following we will represent each goal by means of a symbolic frame reporting the name of the goal, and an informal description of the behaviour of feasibility, opportunity and action functions.

The main goal of the robot is to go to the supermarket and buy some food. This can be expressed as a mandatory sequence of sub-goals:

Name: Main
Relationship: ALL_SEQ
Goal set: Go to supermarket, Pick items, Pay, Go back home

The first sub-goal, *Go to supermarket*, has a feasibility condition implying to check current time in order to ensure that the supermarket is open. Opportunity value is useless, in this case, since, according to the ALL_SEQ semantics, this is the sole sub-goal which can be selected. Its representation is therefore:

Name: Go to supermarket
Feasibility: Is the supermarket open? (check time)
Opportunity: any
Description: Reach the supermarket

Pick items is instead another composite goal and contains a sub-goal for each food type to be brought. We consider that

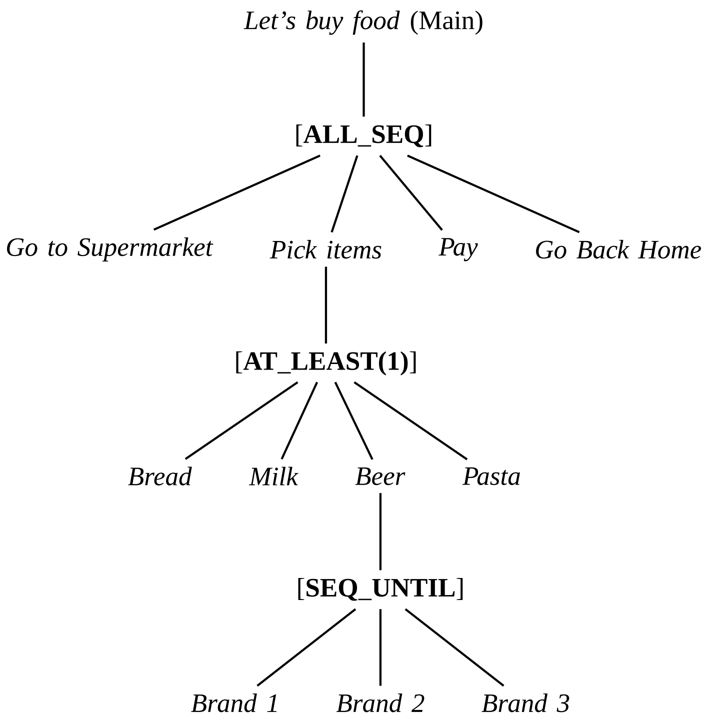

Fig. 2. The behavioural tree of the "food buyer" robot

at least one type of food has to be put in the cart to consider the goal achieved, thus:

Name: Pick items
Relationship: AT_LEAST(1)
Goal set: Break, Milk, Beer, Pasta

Goal *Pay* is a simple one. Its feasibility implies the absence of people in the line and the action requires to pay for the food put in the cart:

Name: Pay
Feasibility: No people in the line?
Opportunity: any
Description: Pay for the food

Last goal of the set, *Go back home*, is quite simple and can be easily described as:

Name: Go back home
Feasibility: True
Opportunity: any
Description: Get the food and go back home

As for the goals related to food picking, *Bread*, *Milk* and *Pasta* are simple and their model is, in practice, the same:

Name: Bread/Milk/Pasta
Feasibility: True
Opportunity: Evaluate distance from the stand
Description: Reach the stand and pick some bread/milk/pasta

Things are different for *Beer* since we expressed a priority preference on brands. For this reason, we model this goal as a composite one using the SEQ_UNTIL relationship:

Name: Beer
Relationship: SEQ_UNTIL
Goal set: Beer brand 1, Beer brand 2, Beer brand 3

Each *Beer brand* n sub-goal can be viewed as simple goal and represented like the ones related to the other food type (i.e. *Bread*, *Milk* and *Pasta*); for this reason, we omit here their description.

V. Related Work

As stated in [8], in which the field of RDEs (Robotic Development Environment) is analyzed by selecting and comparing several different open source RDEs, architectures and programming models for mobile robots have become very important in the last years.

Some previous works, like [15], [16], address the problem of expressing coordination of various concurrent activities in autonomous mobile robots. In [15], an extension of C++, called TDL, is designed for the purpose of syntactically supporting task decomposition, synchronization, execution monitoring, and exception handling. In [16], the authors address the problem of limited physical and computational resources of autonomous robots, which causes various constraints. They present a solution called Task Control Architecture (TCA) that supports the programmer in splitting various type of behaviors onto layered components. Deliberative components handle normal situations, while reactive behaviors handle exceptional one.

In [9], the authors discuss the difficulties related to the practical application of evolutionary approaches to make a mobile robot "learn" a complex behaviour. They propose a modularization of the control architecture to make the robot learn different behaviours by using a task decomposition technique, such that the definition of fitness functions becomes straightforward.

In [7], the problem of automated task planning for service robots is addressed by combining semantic knowledge representation and previous approaches related to AI. A flexible framework is presented, which assists service robots in task planning by means of a semantic ontology constructed for representing environmental description and robot primitive actions. Authors also describe a simple scenario and demonstrate the effectiveness of their work in a simulated environment.

In the field of knowledge processing system for autonomous personal robots, we mention [18] and [19]. Authors of [18] present a knowledge processing system exploiting a first-order knowledge representation based on description logic in order to provide action-centered representation, a way to acquire the grounded concepts and experience by observation and fast inference, which helps autonomous robots to take the right decision in time. In [19] information available in the World Wide Web is used as a knowledge base to help autonomous robots to perform tasks. Web pages are categorized to be used for knowledge processing by autonomous robots and, for this aim, several processing methods are presented.

By the well known DBI (Belief-Desire-Intention) model [13] agent behaviour can be defined in terms of beliefs (the informative component of the system state) , desires (the motivational state of the system), and intentions (encapsulating the deliberative component of the system). On the other hand, our framework focuses on supporting the programmer in developing the deliberative component of robot behaviour through the noted opportunity and feasibility functions.

VI. Conclusions and future work

This paper has presented an abstract framework, called GOLEM, for the design and modelling of behaviour of autonomous robotic systems. In order to give the robot *deliberation abilities*, GOLEM uses a rational approach by organising robot's actions into goals and sub-goals, which must be individually achieved.

The GOLEM framework has been implemented by the authors in two different programming languages. A first implementation is in the C language, specifically optimised for the execution onto platforms based on microcontrollers[3]. This implementation has been used to program the autonomous behaviour of the robots built by the UNICT-TEAM[4] for the Eurobot student robotic competition[5]; it is currently not published on the web but is available on request. The second implementation (called ENIGMA[6]) has been done in the Erlang language and belongs to the authors' research in the field of the application of parallel functional languages to robot programming [14], [17].

Both implementations have been tested on realistic case scenarios, in order to evaluate the effectiveness of the GOLEM approach in real-life situations. Experience showed that the organisation of activities in goals and sub-goals helps a lot the development process, by allowing the programmer to clearly identify the "pieces of the robot's behaviour", treating them as single entities, thus using, also for behaviour programming, the well-known and successful "divide-et-impera" principle.

Detailed description of such implementations, as well as experiences on the use of GOLEM, will be the object of future work.

VII. Acknowledgements

This work has been supported by project PRISMA PON04a2 A/F funded by the Italian Ministry of University.

References

[1] R. A. Brooks, "A robust layered control system for a mobile robot," *Robotics and Automation, IEEE Journal of*, vol. 2, no. 1, pp. 14–23, 1986.

[2] T. S. Chow, "Testing software design modeled by finite-state machines," *IEEE Trans. Software Eng.*, vol. 4, no. 3, pp. 178–187, 1978.

[3] C. Côté, D. Létourneau, F. Michaud, J.-M. Valin, Y. Brosseau, C. Raievsky, M. Lemay, and V. Tran, "Code reusability tools for programming mobile robots," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 2. IEEE, 2004, pp. 1820–1825.

[4] L. Fichera, D. Marletta, V. Nicosia, and C. Santoro, "Flexible robot strategy design using belief-desire-intention model." in *Eurobot Conference*, 2010, pp. 57–71.

[5] G. Fortino, W. Russo, and C. Santoro, "Translating statecharts-based into bdi agents: The dsc/profeta case," in *Multiagent System Technologies*. Springer, 2013, pp. 264–277.

[6] D. Harel, "Statecharts: A visual formalism for complex systems," *Science of computer programming*, vol. 8, no. 3, pp. 231–274, 1987.

[7] Z. Ji *et al.*, "Towards automated task planning for service robots using semantic knowledge representation," in *INDIN*. IEEE, 2012, pp. 1194–1201.

[8] J. Kramer and M. Scheutz, "Development environments for autonomous mobile robots: A survey," *Autonomous Robots*, vol. 22, no. 2, pp. 101–132, 2007.

[9] W.-P. Lee, J. Hallam, and H. H. Lund, "Learning complex robot behaviours by evolutionary computing with task decomposition," in *Learning Robots*. Springer, 1998, pp. 155–172.

[10] D. S. Lees and L. J. Leifer, "A graphical programming language for robots operating in lightly structured environments," in *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*. IEEE, 1993, pp. 648–653.

[11] F. Messina, G. Pappalardo, and C. Santoro, "Integrating cloud services in behaviour programming for autonomous robots," in *Algorithms and Architectures for Parallel Processing*. Springer, 2013, pp. 295–302.

[12] O. Obst, "Specifying rational agents with statecharts and utility functions," in *RoboCup 2001: Robot Soccer World Cup V*, ser. Lecture Notes in Computer Science, A. Birk, S. Coradeschi, and S. Tadokoro, Eds. Springer Berlin Heidelberg, 2002, vol. 2377, pp. 173–182.

[13] A. Rao and M. Georgeff, "BDI agents: From theory to practice," in *Proceedings of ICMAS*. San Francisco, CA, 1995, pp. 312–319.

[14] C. Santoro, "An erlang framework for autonomous mobile robots," in *ERLANG '07: Proceedings of the 2007 ACM SIGPLAN workshop on Erlang*. ACM Press, 2007.

[15] R. Simmons and D. Apfelbaum, "A task description language for robot control," in *IEEE/RSJ*, vol. 3. IEEE, 1998, pp. 1931–1937.

[16] R. G. Simmons, "Structured control for autonomous robots," *Robotics and Automation, IEEE Transactions on*, vol. 10, no. 1, pp. 34–43, 1994.

[17] A. D. Stefano, F. Gangemi, and C. Santoro, "ERESYE: Artificial Intelligence in Erlang Programs," in *ERLANG '05: Proceedings of the 2005 ACM SIGPLAN workshop on Erlang*. New York, NY, USA: ACM Press, 2005, pp. 62–71.

[18] M. Tenorth and M. Beetz, "Knowrobknowledge processing for autonomous personal robots," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*. IEEE, 2009, pp. 4261–4266.

[19] M. Tenorth, U. Klank, D. Pangercic, and M. Beetz, "Web-enabled robots," *Robotics & Automation Magazine, IEEE*, vol. 18, no. 2, pp. 58–68, 2011.

[3]This implementation works in bare metal, without an operating system

[4]http://eurobot.dmi.unict.it

[5]http://www.eurobot.org

[6]https://github.com/ivaniacono/enigma