Dynamic Goal Decomposition and Planning in MAS for Highly Changing Environments

Stefania Costantini and Giovanni De Gasperis

DISIM, Università di L'Aquila, Italy
email: {stefania.costantini, giovanni.degasperis}@univaq.it

Abstract. This paper studies the problem of dynamic goal decomposition and planning in scenarios characterized by a strong inter-dependency between action and context, for instance those related to rescue intervention in a territory upon occurrence of some kind of catastrophic event. We propose an architecture that integrates DALI MASs (DALI Multi-Agent Systems) and ASP (Answer Set Programming) modules for reaching goals in a flexible and timely way, where DALI is a computational-logic-based fully implemented agent-oriented logic programming language and ASP modules allow for affordable and flexible planning capabilities. The proposed DALI MAS architecture exploits these modules for flexible goal decomposition and planning with the possibility to select plans according to a suite of possible preferences and to re-plan upon need. We present a case-study concerning DALI agents which cooperate for exploring an unknown territory under changing circumstances in an optimal or at least sub-optimal fashion. The architecture can be exploited not only by DALI agents, but rather by any kind of logical agent.

1 Introduction and Motivation

Scenarios characterized by **inter-dependency** between action and context are [1] for instance the ones related to the rescue intervention in a territory upon occurrence of some kind of **catastrophic event**. Cities with damaged / nonfunctional roads, people moving to safe places, people in need that cannot move, busy telecommunication channels, chaotic car traffic, etc. In such cases, data and knowledge about the territory is often outdated within a few seconds.

What is required in these scenarios includes:

– repeatedly **monitor** and explore the environment by means of an infrastructure including drones, sensors, robots, and human operators equipped with some kind of system terminals to regain up-to-date knowledge of the environment;
– **suggest** dynamic **plans** of intervention obeying to physical, ethical and organizational constraints (e.g., a sequence of intervention according to priorities);
– **guide** human or robotic rescuers in the exploration of the territory.

Action and context are in such scenarios intertwined and mutually dependent, with actions determined by a context in turn dynamically modified by the actions (e.g., by removal of debris), as well as by external events (like aftershocks in case of an earthquake).

Fig. 1. Top row: *Earthquake*: L'Aquila, Italy, 2009;
Bottom left *Flooding*: USA, 2010;
Bottom right *Fire*: Chile, 2012.

In this paper we propose an approach which seamlessly integrates activity plans and dynamic knowledge acquisition on the environment within the framework of a logic-based multiagent-oriented system. In particular, we concentrate as a case-study on the exploration of the environment: in fact, after catastrophic events previous knowledge about the environment may be no longer valid, and hence the environment must be considered as (at least partially) unknown. Thus, exploration and dynamic acquisition of up-to-date data is a prerequisite for organizing rescue.

Adaptive autonomous agents [2–4] are capable of adapting to partially unknown and potentially changing environments. This requires agents to be capable of various forms of commonsense reasoning and planning. Since [5], we advocated agent architectures capable of smooth integration of several modules/components representing different behaviors/forms of reasoning, possibly based upon different formalism. Therefore, the overall agent's behavior can be seen as the result of dynamic combination of these behaviors, also in consequence of the evolution of the agent's environment.

We propose in particular to adopt Answer Set Programming (ASP) modules, where ASP (cf., among many, [6–9] and the references therein) is a successful logic programming paradigm which is nowadays a state-of-the-art tool for planning and reasoning with affordable complexity, for which many efficient implementations are freely available [10]. We in fact augmented the DALI agent-oriented language, invented and developed by our research group [11–14], with a plugin for the invocation of answer set solvers. ASP modules can be exploited in agents in a variety of ways [5], for instance (but not only) for (limited forms of) reasoning about possibility and necessity. We have

recently enhanced the integration by adopting ASP modules for planning purposes, allowing an agent or a MAS to choose among the various plans that can be obtained by means of suitable preferences. Lately, we also introduced a ProbLog [15] in DALI, to compute answers with associated *probabilities*: in this way, an agent is able to choose the course of action also according to probabilistic considerations.

In this paper, we show an architecture (that we have designed and developed for DALI, but that can be easily adapted to other agent-oriented frameworks) to cope with complex goals, i.e., goals that can take profit from the subdivision into sub-goal because one of the following (or both) is the case:

- the instance size of the planning problem to be solved for reaching the goal is too big for efficient and timely solution, so the instance must be partitioned (if possible) and the sub-solutions must then be combined/merged together;
- the goal naturally splits into sub-goals where plans can/must be devised separately, and then combined/merged together.

The architecture exploits not a single DALI agent but a MAS (Multi-Agent System), with suitable components for planning and executing plans, but also for partitioning goals and controlling the generation/exploitation of solutions, and possible (even partial) re-planning in case of environmental changes.

The effectiveness of this solution is demonstrated by means of a case-study where DALI agents cooperate in order to explore an unknown territory upon occurrence of some kind of catastrophic event (earthquake, fire, flooding, terrorist attack, ect.).

We propose a solution based upon a MAS instead of a monolithic software solution because we consider important that each software component, that we implement as an agent, should partially retain its autonomy during asynchronous event processing. In fact, in this way each agent can be enriched with high-level reasoning/control behaviors that can coexists with the planning/executing activity. The MAS solution also permits to distribute the computational effort and increases overall robustness by means of advanced features such as self-monitoring and self-diagnostic, as shown in [16]. As discussed below, the MAS can be based upon a controller agent which partitions a planning problem and, according to certain features (e.g., related to plan selection), assigns tasks of planning, re-planning and plan execution. ASP modules are meant to be exploited for planning purposes. Qualitative aspects of the proposed solution consist in: (1) the general MAS structure, that can be customized in order to cope with real-world problems rather than toy instances; (2) the interaction between the MAS and the ASP module(s); (3) the adoption of preferences for choosing among possible plans.

The paper is structured as follows. First, we recall ASP and the DALI language and framework. We then present the proposed architecture, and the case study. Finally we discuss the proposal and some related work and conclude.

2   Answer Set Programming in a Nutshell

"Answer set programming" (ASP) is a well-established logic programming paradigm adopting logic programs with default negation under the *answer set semantics*, which [17, 18] is a view of logic programs as sets of inference rules (more precisely, default

inference rules). In fact, one can see an answer set program as a set of constraints on the solution of a problem, where each answer set represents a solution compatible with the constraints expressed by the program. For the theory and the applications of ASP, the reader can refer for instance to [6–9]. Planning is among the most suitable and successful applications of ASP: c.f., e.g., [19, 20] and the references therein, treating planning in ASP even under incomplete information. Several well-developed answer set solvers [10, 6] that compute the answer sets of a given program can be freely downloaded by potential users. The functioning and features of such solvers is illustrated in articles appearing in [6].

Syntactically, an ASP program (or, for short, just "program") Π is a collection of *rules* of the form

$$H \leftarrow L_1, \ldots, L_m, \, not \, L_{m+1}, \ldots, not \, L_{m+n}$$

where H is an atom, $m \geqslant 0$ and $n \geqslant 0$, and each L_i is an atom. Symbol $\leftarrow$ is usually indicated with :- in practical systems. An atom L_i and its negative counterpart $not \, L_i$ are called *literals*. The left-hand side and the right-hand side of the clause are called *head* and *body*, respectively. A rule with empty body is called a *fact*. A rule with empty head is a *constraint*, where a constraint of the form

$$\leftarrow L_1, ..., L_n.$$

states that literals $L_1, \ldots, L_n$ cannot be simultaneously true in any answer set.

Unlike other paradigms, a program may have several answer sets, each of which represents a solution to given problem which is consistent w.r.t. the given problem description and constraints, or may have no answer set, which means that no such solution can be found. Whenever a program has no answer sets, it is said to be say that the program is *inconsistent* (w.r.t. *consistent*). In the case of planning, each answer set (if any exists) represents a plan.

All solvers provide a number of additional features useful for practical programming, that we will introduce only whenever needed. Solvers are periodically checked and compared over well-established benchmarks, and over significant sample applications proposed at the yearly ASP competition (cf. [21] for a recent report).

3   The DALI language: Framework and Applications

DALI [11, 12] (cf. [13] for a comprehensive list of references) is an Agent-Oriented Logic Programming language. The DALI semantics [22] is fully logical, and the change of state proper of agents' functioning is modeled by means of a concept of *Evolutionary Semantics*. DALI agent is triggered by several kinds of events: external events, internal, present and past events.

External events are syntactically indicated by the postfix *E*. Reaction to such events is defined via reactive rules, indicated by the special token :>, i.e., a reactive rule is of the form $External_e vent :> Reaction$; reactive rules can also be defined upon a set of external events. The agent remembers to have reacted by converting an external event

into a *past event* (postfix *P*), with a related time-stamp. An event perceived but not yet reacted to is called "present event" and is indicated by the postfix *N*.

In DALI, **actions** (indicated with postfix *A*) may have or may not have preconditions: in the former case, the actions are defined by actions rules, in the latter case they are just action atoms. An action rule is characterized by the new token :<. Similarly to events, actions are recorded as past actions.

Internal events is the device which makes a DALI agent proactive. An internal event is syntactically indicated by the postfix *I*, and its description is composed of two rules. The first one contains the conditions (knowledge, past events, procedures, etc.) that must be true so that the reaction (in the second rule) may happen. Thus, a DALI agent is able to react to its own conclusions. Internal events are automatically attempted with a default frequency customizable by means of directives in the initialization file, where the frequency will depend upon the very nature of each such event, and the degree of criticality for the agent.

The sample DALI program below is aimed to illustrate how to use the main language features, and which is their role. There is an external event, namely $requestE(K)$, representing some request, say of kind K. The event is first of all detected in the incoming event queue, where at this stage it is considered as a present event $requestN(K)$. The presence of the event determines some kind of reasoning (not specified here), performed by the procedure $evaluate\_request(K, Outcome)$ which will return a result in its output parameter $Outcome$. Then, the event will be reacted to by a dedicated reactive rule, which in the example invokes a procedure $manage\_request(K)$; in general however, the reaction may involve any combination of reasoning tasks and action execution. The $manage\_request(K)$ procedure is supposed to check some conditions, choose a suitable action to perform and then try to actually execute this action by invoking $perform\_actionA(Act)$. However, executing the action may not be immediate, as in this case the action has preconditions (expressed in the last rule). Specifically, for the action to be executable the necessary resources must be acquired. In the meanwhile, if the evaluation of the request leads to an outcome of $suspect\_attack$, then this is interpreted as an internal event, namely $evaluate\_requestI(K, suspect\_attack)$, which leads to another reaction, in this case raising some kind of alert. Thus, application of event-condition-action reactive rules constitutes only a part of a DALI agent behavior. In fact, in addition to reacting to external events a DALI agent can reason on incoming events, can trigger new reactions based upon internal reasoning, can reason about actions, etc.

$evaluate\_request(K, Outcome)$:- $requestN(K)$.
$evaluate\_request(K, Outcome)$:- ...
$evaluate\_requestI(K, suspect\_attack)$:> $raise\_alert(suspect\_attack)$.
$requestE(K)$:> $manage\_request(K)$.

$manage\_request(K)$:- $check\_conditions(K)$,
$\qquad\qquad\qquad choose\_action(Act), perform\_actionA(Act)$.
$perform\_actionA(Act)$:< $get\_resources(Act)$.

The DALI communication architecture [23] implements the DALI/FIPA protocol, which consists of the main FIPA primitives, plus few new primitives which are par-

ticular to DALI. The architecture includes a filter on communication based on special primitives, ontologies and forms of commonsense reasoning. This advanced communication architectures allows DALI agents to implement quite sophisticated forms of interaction, like learning by exchanging sets of rules [24]. DALI has also a *goal* construct to define simple plans, similarly to what done in the AgentSpeak language [25, 26]. As seen below however, for more involved planning tasks one can employ suitable ASP modules.

The DALI programming environment at current stage of development [14] offers a multi-platform folder environment, built upon Sicstus Prolog programs, shells scripts, Python scripts to integrate external applications, a JSON/HTML5/jQuery web interface to integrate into DALI applications, with a Python/Twisted/Flask web server capable to interact with A DALI MAS at the back-end. We have recently devised a cloud DALI implementation, reported in [27, 28]. In fact, as we have since long been convinced of the potential usefulness of the DALI logical agent-oriented programming language in the cognitive robotic domain, in the above-mentioned papers we have presented the extensions to the basic pre-existing DALI implementation which add a number of useful new features, and in particular allow a DALI MAS to interact with robots. As shown in [27], the DALI framework has been extended to "DALI 2.0" by using open sources packages, protocols and web based technologies. DALI agents can thus be developed to act as high level cognitive robotic controllers, and can be automatically integrated with conventional embedded controllers. The web compatibility of the framework allows real-time monitors and graphical visualizers of the underline MAS activity to be specified, for checking the interaction between an agent and the related robotic subsystem. The cloud package ServerDALI allows a DALI MAS to be integrated into any practical environment. In [28] illustrate the recent "Koiné DALI" framework, where a Koiné DALI MAS can cooperate without problems with other MASs, programmed in other languages (logical or non-logical), and with object-oriented applications. In summary, the enhanced DALI can be used for multi-MAS applications and hybrid multi-agents and object-oriented applications, and can be easily integrated into preexisting applications.

The DALI framework has been experimented, e.g., in applications for user monitoring and training, in emergencies management (like first aid triage assignment), in security or automation contexts, like home automation or processes control, and, more generally, in every situation that is characterized by events (either simple events and/or events that are correlated to other ones even in complex patterns). An architecture encompassing DALI agents and called F&K (Friendly-and-Kind) system [29] has been proposed for (though not restricted to) applications the e-Health domain. F&Ks are "knowledge-intensive" systems, providing flexible access to dynamic, heterogeneous, and distributed sources of knowledge and reasoning, within a highly dynamic computational environment consisting of computational entities, devices, sensors, and services available in the Internet and in the cloud. As a suitable general denomination for systems such as F&Ks we propose "Dynamic Proactive Expert Systems" (DyPES): in fact, such systems are aimed at supporting human experts and personnel or human users in a knowledgeable fashion, so they are reminiscent of the role of traditional expert systems. However, they are proactive in the sense that such systems have objectives (e.g.,

monitoring patients, managing resources, exploring territories, etc.) that they pursue autonomously, requiring human intervention only when needed. They are also dynamic, because they are able to exploit not only a predefined knowledge base: rather, they are equipped with a number of reasoning modules, and they are able to locate other such modules, and the necessary knowledge and reasoning auxiliary resources. In fact, DyPESs are characterized by "Knowledge Intensity", in the sense that in general a large amount of heterogeneous information and data must be retrieved, shared and integrated in order to reason within the system's domain. DyPESs can be Cyber-Physical Systems integrating software and physical components [30], and can be able to perform Complex Event Processing, i.e., to actively monitor event data so as to make automated decisions and take time-critical actions (DALI has been in fact empowered with CEP capabilities [31]).

Agents (and in particular robotic agents) have complex goals that may need to be decomposed, either hierarchically or anyway into related sub-goals; moreover, such goals may change in time depending upon the interaction with the environment. Prolog-based logical agents such as DALI agents but also agents written in other agent-oriented computational-logic-based languages (e.g., AgentSpeak [25, 26], GOAL [32, 33], 3APL [34, 35]) can devise and execute plans. However, they are not easily able to decompose goals into sub-goals, evaluate (based upon preferences) alternative plans, and re-plan if needed, possibly for some sub-goals only; implementing such features within a single agent would in fact make the agent code heavy to understand and execute.

We have since long equipped DALI with a plugin for invoking ASP solvers and thus executing ASP modules. If such a module is used for planning, the possibility has been recently introduced to choose among the generated plans based upon preferences; the preference strategies implemented so far are: (i) shortest plan; (ii) minimal-cost plan; (iii) plan including a minimum/maximum number of a certain kind of actions; we intend to implement plan evaluation based upon preferences on resource consumption, following the principles of [36–39].

Below we propose a DALI MAS architecture aimed at goal decomposition, sub-goal assignment, planning and re-planning concerning complex goals.

4   The MAS Architecture

In this section we illustrate the features of the proposed architecture. The DALI MAS is intended to fulfill the so-called *bounded rationality principle*, by which a plan for reaching a goal has to be devised and executed in a timely manner before a ultimate T_{max} deadline. Consequently, there is a second deadline $T_{PlanMax} < T_{Max}$ by which a plan has to be computed and selected, so that the remaining time is sufficient for plan execution.

Thus, given the input set $T_{PlanMax}, T_{Max}, G, N$, where G is the goal, N is the instance size of the problem to be solved (if applicable), the MAS operates via the following steps.

 (i) Decompose the overall goal into suitable sub-goals;
(ii) For each sub-goal, generate a plan within the $T_{PlanMax}$ deadline;

(iii) Execute the plan within the T_{Max} deadline; in case of failure (insufficient time), maximize the length of the partially executed plan;

(iv) In case of a change of conditions in the environment, re-plan, possibly limiting this activity to specific sub-goals resulting from the partitioning.

Sub-goals can be determined by any kind of goal partitioning algorithm. In the disaster management case study, here discussed, it is obtained simply by sub-dividing the main geographical area into slightly overlapping sub-territories. Other algorithms can be adopted to generate sub-plans, for instance those presented in [40], [41] and [42]. The planner agent equipped with an ASP module may find more than one plan for each (sub-)goal; so it is useful (as said before) to apply metrics by which a plan could be preferred to another one. The proposed DALI MAS architecture is shown in Figure 2 and the agent behaviors are the following.
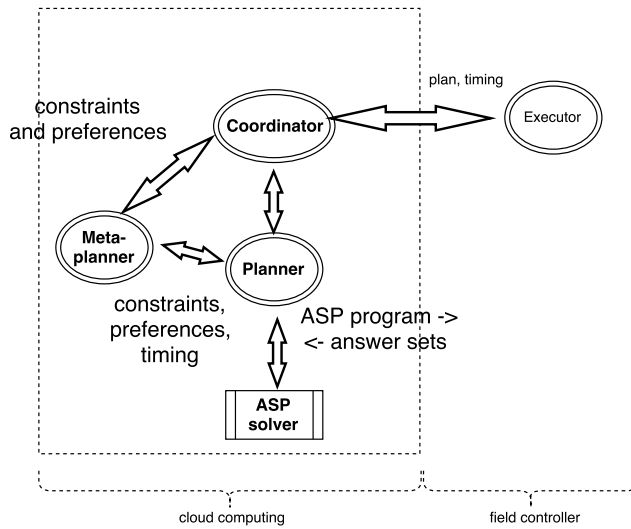

Fig. 2. DALI MAS architecture: Coordinator, Meta-Planner, Planner, Explorer agents. The MAS can be deployed over a cloud computing architecture, thus distributing and balancing the required computational resources. The ASP module is executed via an external solver, configurable depending on the required capabilities. The EXPLORER agent is supposed to execute a plan, possibly working "in the field", i.e., embedded in a mobile robot or some other facility.

– **COORDINATOR** agent: this agent synchronizes all the actions of the MAS and updates the global state of goal solving. Its tasks are the following.

(a) Ensure the proper activation of the MAS.

(b) Interact with the external world and whenever needed set new objectives for the MAS or revise the present goals.

(c) Initialize the $T_{PlanMax}$ and T_{Max} deadlines, depending upon the goal.

(d) Decompose a goal into subgoals.

(e) For each subgoal, activate a copy of the **META-PLANNER** agent, possibly providing as input the preference criterion for plan selection.

(f) receive from the **META-PLANNER** agent the plan to be executed up to $T_{PlanMax}$ and deliver the plan to the **EXPLORER** agent, which is in charge of plan execution within maximum time T_{Max} - $T_{PlanMax}$.

(h) If time elapses, or new events occur, cancel the current running plan and if applicable send a replan indication to the **META-PLANNER**.

(h) Logs all events to a log server.

- **META-PLANNER** agent, whose tasks are the following.

  (a) Receive the triggering event from the **COORDINATOR** to start the search for a new plan.

  (b) Generate input for the **PLANNER** agent while monitoring its performances. If the **PLANNER** agent does not deliver before $T_{PlanMax}$, cancel the plan request and ask **PLANNER** to generate a trivial plan.

  (c) Apply plan selection accorded to preferences, either local or set by **COORDINATOR** agent. That is, it exploits the given preference criteria in order to select the plan which is closer to the specified preferences whenever the **PLANNER** returns more than one answer.

  (c) If requested by **COORDINATOR** agent, ask **PLANNER** for re-planning with updated input.

- **PLANNER** agent, which receives as input the time constraints $T_{PlanMax}, T_{Max}$ and spatial constraints $C_\%, N, F$ (e.g.: coverage, number of reach/do not reach cells) from **META-PLANNER** and generates possible plans via an ASP module, if possible within the $T_{PlanMax}$ deadline. If more than a single answer is produced by the ASP solver, it returns all available plans to the **META-PLANNER**. If no solution exists, it generates a trivial plan (if possible), i.e. a simple greedy algorithm without global optimization.

- **EXPLORER**: puts into action the plan provided by the **COORDINATOR**, if possible within the T_{Max} deadline, and notifies the **COORDINATOR** upon completion. The explorer is in charge of plan execution and is so-called as a reminiscence of the case study presented below; in general however, the name is justified because this agent can execute plans (also) by means of physical components in a Cyber-Physical System, and/or by means of robotic elements of various kinds. In Figure 1, **EXPLORER** is designated as "field controller" as plan execution is situated into some environment.

5 Case Study

The architecture presented above has been inspired and motivated by a case-study that has been actually implemented and experimented, and presented in [43]. The overall goal in the case study is to explore an unknown territory upon occurrence of some kind of catastrophic event (earthquake, fire, flooding, terrorist attack, ect.). For simplicity, we have modeled the territory (also called "area") as a set of a $N * N$ parts represented as

chessboards, i.e., squares of cells, where some cells are marked as unreachable/forbidden, and are therefore considered as "holes" in the chessboard. This represents the fact that the agents may be notified by an external authority or by other sources of the actual impossibility of traversing that location because of some kind of obstruction/danger. The forbidden/unreachable locations can change in time.

For the sake of experiments, the robot that each agent employs for exploration has been represented (in the case study) as a chess' knight piece, which performs knight leaps. This is to signify that a real robot (whatever its kind) will in practice have limited possibilities of movement. In this way, the problem of exploration of a single piece of territory can be modeled as a variant of the well-known "knight tour with holes" problem, for which well-known ASP solutions exist. The ultimate objective would be that of devising an Hamiltonian path, thus fully exploring the given piece of territory while skipping the forbidden squares. As however the Hamiltonian path option results too heavy with reasonable instance size (actually, it is too heavy for size more than 8), we resorted to sub-optimal solutions which adopt soft constraints in order to visit each square as few times as possible.

The Knight Tour with holes problem has constituted a benchmark in recent ASP competitions, aimed at comparing ASP solvers performances. We performed a number of modifications to the original version [44] concerning: the representation of holes; the objective of devising a path which, though not Hamiltonian, guarantees a required degree of coverage with the minimum number of multiple-traversals; simple forms of loop-checking for avoiding at least trivial loops. For the sake of completeness, below is the sketch of our solution, formulated for the DLV ASP solver [45], though it might be easily reformulated for other solvers. The key modifications to the base solution are the following.

– We modified the *reached* constraint, and transformed it into a "soft" (or *weak*) constraint, satisfied if possible, denoted by connective :∼, so as not to be forced to finding a Hamiltonian path.

```
reached(X,Y) :- move(1,1,X,Y).
reached(X,Y) :- reached(X1,Y1),
   move(X1,Y1,X,Y).
:∼ cell(X,Y),
      not forbidden(X,Y),
      not reached(X,Y).
```

– We added a coverage-satisfaction rule, where *coverage* denotes the required degree of coverage and $number\_forbidden$ the number of holes, and V is the instance size, i.e., the chessboard edge. The maximum possible coverage is 100% of the available cells, i.e., $M = V * V$, while the minimum coverage N is computed in terms of *coverage*, considering the holes. Suitable application of the *count* DLV constraint [45] guarantees the desired coverage.

```
coverage(95). % sample coverage degree,
              % can be changed
number_forbidden(5).
cov(N) :-
  N <= #count{X,Y : reached(X,Y)} <= M,
```

```
size(V), coverage(Z),
number_forbidden(F),
M = V * V, N2 = M * Z,
N3 = N2 /100, N = N3 - F.
```

Experimental results, presented in [43, 46], have demonstrated the usefulness of the proposed MAS architecture, that is actually able to effectively cope with real-world instance sizes. The architecture in this case study works as follows.

- The COORDINATOR agent partitions the territory that must be explored into a number of (possibly overlapping) sections (chessboards) of reasonable size, each one to be assigned to a META-PLANNER instance.
- Each instance of the META-PLANNER agent exploits its own associated copy of the planner agent (ASP module), which computes possible plans.
- Different preference policies for plan selection can possibly be associated with different sections of the territory to be explored, and can therefore be applied to the associated META-PLANNER, according to directions provided by the user/environment.
- Each plan selected for execution (exploration to be performed) is assigned to a separate EXPLORER agent, specifically assigned to that territory section.
- The COORDINATOR will devise re-planning for each portion of the territory for which the unreachable location have changed.

Reasonable metrics to evaluate plans returned by the ASP module can be in terms of: (i) number of cells that have to be visited when using coverage, (ii) length of the path, (iii) presence of loops (when the Hamiltonian constraint is released), (iv) plan cost, in case there is a specific cost associated to each cell, (v) preference on which (kinds of) cells one should prioritarily visit. Preference criteria can then be defined by selecting one metric, or by combining different metrics: for instance, a criterium may consist in preferring the shortest path, if it does not exceed a certain cost.

6 Related Work, Discussion and Concluding Remarks

Goal reasoning and goal decomposition is very relevant task, crucial in dynamic and unpredictable environments where such decomposition can hardly be performed "a priori". We have proposed a MAS architecture for flexible goal decomposition, plan formation and execution [1]. In real applications, a MAS for each (class of) goal(s) would be designed, implemented and located into the DALI cloud. In fact, each MAS will be programmed according to a goal to be reached, i.e., to a problem to be solved. Then, an agent that needs to solve a certain goal can refer to the suitable MAS. As mentioned,

---

[1] Previous versions of this paper have been presented at SIRLE 2018, AAAI 2018 Spring Symposium on Integrating Representation, Reasoning, Learning, and Execution for Goal Directed Autonomy, held at Stanford University, Palo Alto, CA, United States, March 26-28, 2018 and at the 6th Goal Reasoning Workshop at IJCAI 2018 (both with no archival Proceedings).

the DALI framework allows uniform access also to agents written in other languages/-formalisms. So, the proposed solution is not DALI-specific but rather can be generally adopted.

Relevant related work exists concerning robotic applications in changing environment. The work presented in [47] proposes a method to temporarily reassign individual service robots in a multi-robot system from their original task provide guide to humans in need from one location to another in the environment. This is related to rescue after some kind of unexpected event. Our approach comes prior to rescue, of which it may constitute a prerequisite: in fact, after catastrophic events previous knowledge of the environment may become obsolete, so that the action of service robots can become difficult or even impossible. Thus, exploration can provide such a multi-robot system with reliable information (to be constantly updated) to perform their task.

[48, 49] propose the ER-DCOPs approach to model DCOPs (Distributed Constraint Optimization Problems, that are useful to model a number of multi-agent coordination problems) when there is uncertainty in constraint utilities. The ER-DCOPs solution can implemented in ASP (a highly parallel GPU-based implementation is also described), so as to exploit reasoning in order to reduce the solution search space. Agents are supposed to communicate in order to jointly solve the given problem. This kind of solution can again be very useful for rescue and assistance, in order, for instance to decide to which hospitals to send the injured and which route to advise to the survivors so as to avoid bottlenecks. For exploration, we have devised instead a solution that for the sake of efficiency makes interaction among the meta-planners and the explorer agents unnecessary, even if this implies to disregard full optimality.

7   Acknowledgement

References

1. Costantini, S., De Gasperis, G., Nazzicone, G., Tarantino, L.: Context and action: A unitary vision within a logic-based multi-agent environment. Lecture Notes in Information Systems and Organisation **18** (2016) 97–111
2. Fisher, M., Bordini, R.H., Hirsch, B., Torroni, P.: Computational logics and agents: a road map of current technologies and future trends. Computational Intelligence Journal **23**(1) (2007) 61–91
3. Bordini, R.H., Braubach, L., Dastani, M., Fallah-Seghrouchni, A.E., Gómez-Sanz, J.J., Leite, J., O'Hare, G.M.P., Pokahr, A., Ricci, A.: A survey of programming languages and platforms for multi-agent systems. Informatica (Slovenia) **30**(1) (2006) 33–44
4. Artikis, A., Pitt, J.V.: Specifying open agent systems: A survey. In Artikis, A., Picard, G., Vercouter, L., eds.: Engineering Societies in the Agents World IX, 9th International Workshop, ESAW 2008, Revised Selected Papers. (2008) 29–45
5. Costantini, S.: Answer set modules for logical agents. In de Moor, O., Gottlob, G., Furche, T., Sellers, A., eds.: Datalog Reloaded: First International Workshop, Datalog 2010. Volume 6702 of LNCS. Springer (2011) Revised selected papers.

6. Brewka, G., Eiter, T., (eds.), M.T.: Answer set programming: Special issue. AI Magazine **37**(3) (2016)

7. Baral, C.: Knowledge representation, reasoning and declarative problem solving. Cambridge University Press (2003)

8. Leone, N.: Logic programming and nonmonotonic reasoning: From theory to systems and applications. In Baral, C., Brewka, G., Schlipf, J., eds.: Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007. (2007)

9. Truszczyński, M.: Logic programming for knowledge representation. In Dahl, V., Niemelä, I., eds.: Logic Programming, 23rd International Conference, ICLP 2007. (2007) 76–88

10. Web-references: Some ASP solvers Clasp: `potassco.sourceforge.net`; Cmodels: `www.cs.utexas.edu/users/tag/cmodels`; DLV: `www.dbai.tuwien.ac.at/proj/dlv`; Smodels: `www.tcs.hut.fi/Software/smodels`.

11. Costantini, S., Tocchio, A.: A logic programming language for multi-agent systems. In Flesca, S., Greco, S., Leone, N., Ianni, G., eds.: Logics in Artificial Intelligence, European Conference, JELIA 2002, Proceedings. Volume 2424 of Lecture Notes in Computer Science., Springer (2002)

12. Costantini, S., Tocchio, A.: The DALI logic programming agent-oriented language. In Alferes, J.J., Leite, J.A., eds.: Logics in Artificial Intelligence, 9th European Conference, JELIA 2004, Proceedings. Volume 3229 of Lecture Notes in Computer Science., Springer (2004) 685–688

13. Costantini, S.: The DALI agent-oriented logic programming language: Summary and references 2015 (2015)

14. De Gasperis, G., Costantini, S., Nazzicone, G.: Dali multi agent systems framework, doi 10.5281/zenodo.11042. DALI GitHub Software Repository (July 2014) DALI: `http://github.com/AAAI-DISIM-UnivAQ/DALI`.

15. Bruynooghe, M., Mantadelis, T., Kimmig, A., Gutmann, B., Vennekens, J., Janssens, G., Raedt, L.D.: Problog technology for inference in a probabilistic first order logic. In: ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings. (2010) 719–724

16. Bevar, V., Muccini, H., Costantini, S., Gasperis, G.D., Tocchio, A.: A multi-agent system for industrial fault detection and repair. In: Advances on Practical Applications of Agents and Multi-Agent Systems. Advances in Intelligent and Soft Computing, Springer, Berlin Heidelberg (2012) 47–55 Paper and demo.

17. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In Kowalski, R., Bowen, K., eds.: Proceedings of the 5th International Conference and Symposium on Logic Programming (ICLP/SLP'88). The MIT Press (1988) 1070–1080

18. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing **9** (1991) 365–385

19. Son, T.C.: Answer set programming and its applications in planning and multi-agent systems. In Balduccini, M., Janhunen, T., eds.: Logic Programming and Nonmonotonic Reasoning - 14th International Conference, LPNMR 2017, Proceedings. Volume 10377 of Lecture Notes in Computer Science., Springer (2017) 23–35

20. Romero, J., Schaub, T., Son, T.C.: Generalized answer set planning with incomplete information. In Bogaerts, B., Harrison, A., eds.: Proceedings of the 10th Workshop on Answer Set Programming and Other Computing Paradigms co-located with the 14th International Conference on Logic Programming and Nonmonotonic Reasoning, ASPOCP@LPNMR 2017. Volume 1868 of CEUR Workshop Proceedings., CEUR-WS.org (2017)

21. Calimeri, F., Ianni, G., Krennwallner, T., Ricca, F.: The answer set programming competition. AI Magazine **33**(4) (2012) 114–118

22. Costantini, S., Tocchio, A.: About declarative semantics of logic-based agent languages. In Baldoni, M., Endriss, U., Omicini, A., Torroni, P., eds.: Declarative Agent Languages and Technologies III, Third International Workshop, DALT 2005, Selected and Revised Papers. Number 3904 in Lecture Notes in Computer Science. Springer (2006) 106–123

23. Costantini, S., Tocchio, A., Verticchio, A.: Communication and trust in the DALI logic programming agent-oriented language. Intelligenza Artificiale **2**(1) (2005) 39–46 Journal of the Italian Association AI*IA.

24. Costantini, S., Tocchio, A.: Learning by knowledge exchange in logical agents. In Corradini, F., Paoli, F.D., Merelli, E., Omicini, A., eds.: WOA 2005: Dagli Oggetti agli Agenti. 6th AI*IA/TABOO Joint Workshop "From Objects to Agents": Simulation and Formal Analysis of Complex System, Pitagora Editrice Bologna (2005) 1–8

25. Rao, A.S., Georgeff, M.: Modeling rational agents within a BDI-architecture. In: Proceedings of the Second Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'91), Morgan Kaufmann (1991) 473–484

26. Bordini, R.H., Hübner, J.F.: Semantics for the Jason variant of AgentSpeak (plan failure and some internal actions). In Coelho, H., Studer, R., Wooldridge, M., eds.: ECAI 2010 - 19th European Conference on Artificial Intelligence, Proceedings. Volume 215 of Frontiers in Artificial Intelligence and Applications., IOS Press (2010) 635–640

27. Costantini, S., De Gasperis, G., Nazzicone, G.: DALI for cognitive robotics: Principles and prototype implementation. In Lierler, Y., Taha, W., eds.: Practical Aspects of Declarative Languages - 19th International Symposium, Proceedings. Volume 10137 of Lecture Notes in Computer Science., Springer (2017) 152–162

28. Costantini, S., De Gasperis, G., Pitoni, V., Salutari, A.: Dali: A multi agent system framework for the web, cognitive robotic and complex event processing. In: Proceedings of the 32nd Italian Conference on Computational Logic. Volume 1949 of CEUR Workshop Proceedings., CEUR-WS.org (2017) 286–300 http://ceur-ws.org/Vol-1949/CILCpaper05.pdf.

29. Aielli, F., Ancona, D., Caianiello, P., Costantini, S., Gasperis, G.D., Marco, A.D., Ferrando, A., Mascardi, V.: FRIENDLY & KIND with your health: Human-friendly knowledge-intensive dynamic systems for the e-health domain. In: Highlights of Practical Applications of Scalable Multi-Agent Systems. The PAAMS Collection - International Workshops of PAAMS 2016, Proceedings. Volume 616 of Communications in Computer and Information Science., Springer (2016) 15–26

30. Khaitan, S.K., McCalley, J.D.: Design techniques and applications of cyberphysical systems: A survey. IEEE Systems Journal **9**(2) (2015) 350–365

31. Costantini, S.: Ace: a flexible environment for complex event processing in logical agents. In Matteo Baldoni, L.B., Dastani, M., eds.: Engineering Multi-Agent Systems, Third International Workshop, EMAS 2015, Revised Selected Papers. Volume 9318 of Lecture Notes in Computer Science., Springer (2015)

32. Hindriks, K.V.: Programming rational agents in goal. In: Multi-Agent Programming. Springer US (2009) 119–157

33. Hindriks, K.: A verification logic for goal agents. In Dastani, M.M., Hindriks, K., Meyer, J.J.C., eds.: Specification and Verification of Multi-agent Systems. Springer (2010)

34. Dastani, M., van Riemsdijk, B., Dignum, F., Meyer, J.J.C.: A programming language for cognitive agents goal directed 3apl. In Dastani, M., Dix, J., Fallah-Seghrouchni, A.E., eds.: Programming Multi-Agent Systems, First International Workshop, PROMAS 2003, Selected Revised and Invited Papers. Volume 3067 of Lecture Notes in Computer Science., Springer (2004) 111–130

35. Dastani, M., van Riemsdijk, M.B., Meyer, J.J.C.: Programming multi-agent systems in 3APL. In Bordini, R.H., ed.: Multi-Agent Programming: Languages, Platforms and Applications. Volume 15 of Multiagent Systems, Artificial Societies, and Simulated Organizations. Springer (2005) 39–67

36. Costantini, S., Formisano, A.: Answer set programming with resources. Journal of Logic and Computation **20**(2) (2010) 533–571

37. Costantini, S., Formisano, A.: Modeling preferences and conditional preferences on resource consumption and production in ASP. Journal of of Algorithms in Cognition, Informatics and Logic **64**(1) (2009)

38. Costantini, S., Formisano, A.: Weight constraints with preferences in ASP. In Delgrande, J.P., Faber, W., eds.: Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, Proceedings. Volume 6645 of Lecture Notes in Computer Science., Springer (2011) 229–235

39. Costantini, S., Formisano, A., Petturiti, D.: Extending and implementing RASP. Fundam. Inform. **105**(1-2) (2010) 1–33

40. Shivashankar, V., Alford, R., Kuter, U., Nau, D.S.: The GoDeL planning system: A more perfect union of domain-independent and hierarchical planning. In: IJCAI. (2013) 2380–2386

41. Alford, R., Shivashankar, V., Roberts, M., Frank, J., Aha, D.W.: Hierarchical planning: Relating task and goal decomposition with task sharing. In: IJCAI. (2016) 3022–3029

42. Cox, M.T., Dannenhauer, D., Kondrakunta, S.: Goal operations for cognitive systems. In: AAAI. (2017) 4385–4391

43. Costantini, S., De Gasperis, G., Nazzicone, G.: Exploration of unknown territory via DALI agents and ASP modules. In Omatu, S., Malluhi, Q.M., Rodríguez-González, S., Bocewicz, G., Bucciarelli, E., Giulioni, G., Iqba, F., eds.: Distributed Computing and Artificial Intelligence, 12th International Conference, DCAI 2015, Salamanca, Spain, June 3-5, 2015. Volume 373 of Advances in Intelligent Systems and Computing., Springer (2015) 285–292

44. Calimeri, F., Zhou, N.F.: Knight tour with holes ASP encoding (2014) See `http://www.mat.unical.it/aspcomp2013/files/links/benchmarks/encodings/aspcore-2/22-Knight-Tour-with-holes/encoding.asp`.

45. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The dlv system for knowledge representation and reasoning. ACM Transactions on Computational Logic **7**(3) (2006) 499–562

46. Nazzicone, G.: Definizione, implementazione e sperimentazione di estensioni del linguaggio orientato agli agenti DALI nel campo della robotica cognitiva (Definition, implementation and experimentation of extensions to the DALI agent-oriented language for applications in cognitive robotics). PhD thesis, Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica, Università degli Studi di L'Aquila (2018) in Italian, Supervisor Prof. Stefania Costantini.

47. Zhang, S., Jiang, Y., Sharon, G., Stone, P.: Multirobot symbolic planning under temporal uncertainty. In Larson, K., Winikoff, M., Das, S., Durfee, E.H., eds.: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017. (2017) 501–510

48. Le, T., Son, T.C., Pontelli, E., Yeoh, W.: Solving distributed constraint optimization problems using logic programming. In Bonet, B., Koenig, S., eds.: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, AAAI Press (2015) 1174–1181

49. Le, T., Fioretto, F., Yeoh, W., Son, T.C., Pontelli, E.: Er-dcops: A framework for distributed constraint optimization with uncertainty in constraint utilities. In Jonker, C.M., Marsella, S., Thangarajah, J., Tuyls, K., eds.: Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent System, ACM (2016) 606–614