Flexible Goal-Directed Agents' Behavior via DALI MASs and ASP Modules

Stefania Costantini, Giovanni De Gasperis
Dip. di Ingegneria e Scienze dell'Informazione e Matematica (DISIM),
Università di L'Aquila, Coppito I-67100, L'Aquila, Italy
email: {stefania.costantini, giovanni.degasperis}@univaq.it

Abstract

This paper describes the architecture that integrates DALI MASs (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 includes solvers that allow affordable and flexible planning capabilities. The proposed DALI MAS architecture exploits such 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 an abstract 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.

Introduction

Adaptive autonomous agents are capable of adapting to partially unknown and potentially changing environments (Knudson and Tumer 2011), (Jiming 2001). This requires agents to be capable of various forms of commonsense reasoning and planning over a distributed multi agent architecture. A related work based on procedural reasoning system and belief desire intention (BDI) architecture is PROPHETA (Fichera et al. 2017), an object oriented procedural Python-based multi agent framework with a declarative language approach, used to control autonomous robots. Since (Costantini 2011), we advocated agent architectures capable of smooth integration of several modules/components representing different behaviors/forms of reasoning, possibly based upon different formalisms. 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 proposed in particular to adopt Answer Set Programming (ASP) modules, where ASP (cf., among many, (Baral 2003; Leone 2007; Truszczyński 2007) and the references therein) is a successful logic programming paradigm suitable for planning and reasoning with affordable complexity; many efficient implementations of ASP solvers are freely available like: CLASP (Gebser et al. 2007), Cmodels (Lierler 2005), DLV (Leone et al. 2006b), Smodels (Elkabani, Pontelli, and Son 2004) . The DALI agent-oriented language and framework was invented, designed and developed in our research group (De Gasperis, Costantini, and Nazzicone 2014; Costantini and Tocchio 2002; 2004; Costantini 2015a); the framework has been lately augmented with a plugin for the invocation of answer set solvers so to build specific modules. The ASP modules can be exploited in agents in a variety of ways: for instance in the case of reasoning about possibility and necessity, and a greater set of reasoning contexts. 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.

In this paper, we show an architecture based on DALI and ASP modules to cope with complex goals, but that can be easily generalized to other agent-oriented frameworks; goals that can take profit from the subdivision into subgoals if one of the following (or both) conditions as met:

- the instance size of the planning problem to be solved for reaching the goal is too big for efficient and timely solution, the instance can be partitioned into sub-problems and the sub-solutions can and must be re-combined/merged together;
- the goal naturally splits into sub-goals where plans can/must be devised separately, and then re-combined/merged together at a later stage.

The architecture exploits not a single DALI agent but a distributed MAS (Multi-Agent System), with suitable components for generating and executing plans; it allows to distribute goals and sub-goals while controlling the generation/exploitation of solutions, and possible (even partial) re-planning in case of environmental changes.

We introduce an ideal case study to show how DALI agents can cooperate in order to explore an unknown territory, such as what can happen in the real world upon occurrence of some kind of catastrophic-like disruptive events (earthquake, fire, flooding, terrorist attack), were geo-localized information can easily become obsolete in few seconds and rescue planning is needed, no matter what is the difficulty.

We propose a solution based upon a MAS instead of a

monolithic software solution because we consider important that each software component, i.e. agent, should partially retain its autonomy during asynchronous event processing, in the context of agent-oriented software engineering methodologies (Gomez-Sanz and Fuentes-Fernández 2015) . 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 among cloud computing facilities and embedded computers so to increase overall robustness by means of advanced features such as self-monitoring and self-diagnostic, as shown in (Bevar et al. 2012). As discussed below the MAS can be based upon a controller agent which partitions a planning problem, established 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; (2) the interaction between the MAS and the ASP module(s); (3) the adoption of user preferences for choosing among possible plans.

The paper is structured as follows. In the first two sections we recall ASP and the DALI language and framework. We then present the proposed MAS architecture, and an abstract case study. Finally we discuss the proposal and conclude.

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 (Gelfond and Lifschitz 1988; 1991) 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 applications of ASP, the reader can refer for instance to (Baral 2003; Leone 2007; Truszczyński 2007). However, planning is among the more suitable an successful applications of ASP , cf (Son 2017; Romero, Schaub, and Son 2017) and the references therein, were planning in ASP is analyzed even under incomplete information.

Syntactically, a 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 negated 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 a conventional logic program, a ASP program may have several answer sets, each of which represent a consistent solution to given problem and constraints, or may have no answer set at all, which means that no solution can be found. Whenever a program has no answer sets, it is said 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 challenging sample applications proposed at the yearly ASP competition (cf. (Calimeri et al. 2012), (Gebser, Maratea, and Ricca 2016) for recent reports).

The DALI language: Framework and Applications

DALI (Costantini and Tocchio 2002; 2004) is an Agent-Oriented Logic Programming language, (Costantini 2015a) for a comprehensive and updated list of references. A DALI agent is triggered by several kinds of asynchronous events: external events, internal, present and past events. A DALI MAS does not explicitly requires using a global clock mechanism, but temporal logic can be implemented inside agents.

External events are syntactically indicated by the postfix *E*. Reaction to each such event is defined by a reactive rule, where the special token :>. The agent remembers to have reacted by converting an external event into a *past event* (postfix *P*). 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 not 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 what makes a DALI agent 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 internal frequency customizable by means of directives in the agent initialization file, where the frequency will depend upon the very nature of each such event, and the degree of criticality for the agent.

The DALI communication architecture implements the DALI/FIPA protocol (Foundation for Intelligent Physical Agents 2003), which consists of the main FIPA primitives, plus few new primitives which are particular to DALI. The architecture may also include a filter on communication based on ontologies and forms of commonsense reasoning, as shown in previous works.

The DALI programming environment at current stage of development (De Gasperis, Costantini, and Nazzicone 2014) offers a multi-platform folder environment, built upon Sicstus Prolog programs, shell scripts, Python scripts to integrate external applications, a JSON/HTML5/jQuery web user interface to integrate into DALI applications, with a Python/Twisted/Flask web server capable to interact with A DALI MAS at the backend. We have recently devised a cloud DALI implementation, reported in (Costantini, De

Gasperis, and Nazzicone 2017; Costantini et al. 2017). 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 abovementioned papers we have presented the extensions to the basic pre-existing DALI implementation with a number of useful new features, and in particular allow a DALI MAS to interact with robots over messages buses like ROS, YARP, Redis event broker. As shown in (Costantini, De Gasperis, and Nazzicone 2017), 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 (Costantini et al. 2017) paper we have illustrated the new "Koiné DALI" framework, where a Koiné DALI MAS can cooperate without problems with other MASs, programmed in other languages, 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 preexistent 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 asynchronous 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 (Aielli et al. 2016) has been proposed for (though not restricted to) applications the eHealth 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 (Khaitan and McCalley 2015), 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 (Costantini 2015b)).

Agents (and in particular robotic agents) have complex goals that may need to be decomposed, either hierarchically or anyway into related subgoals; 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 (Rao and Georgeff 1991; Bordini and Hübner 2010), GOAL (Hindriks 2009; 2010), 3APL (Dastani et al. 2004; Dastani, van Birna Riemsdijk, and Meyer 2005)) can devise and execute plans. However, they are not easily able to decompose goals into subgoals, evaluate (based upon preferences) alternative plans, and re-plan if needed, possibly for some subgoals 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. When this module is used for planning, it would be possible to choose among the generated plans based upon qualitative and quantitative user 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 (Costantini and Formisano 2010; 2009; Costantini, Formisano, and Petturiti 2010).

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

The ASP-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* (Gigerenzer 2004), which we translate that a plan for reaching a goal shall 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 to execute that plan. Parameters $T_{PlanMax}$ and T_{Max} are indeed dependent of the problem domain. At the current state of development they have to be determined by the MAS-ASP designer and stay constant always during run-time phase.

We also consider the hypothesis that for each problem P proposed to the MAS, a trivial solution plan can always be computed in time T_{Pt} by using a well tested deterministic algorithm, such that T_{Pt} is a negligible time compared to T_{Ps}, which is the minimum time needed to generate an acceptable sub-optimal plan.

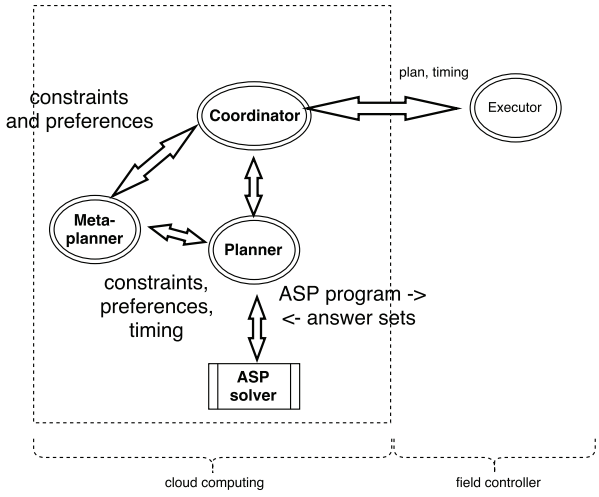

Figure 1: DALI ASP-MAS architecture: **Coordinator**, **Meta-Planner**, **Planner**, **Executor agents**. The MAS can be deployed over a cloud computing architecture, thus distributing and balancing the required computational resources. The ASP module is executed via an external solver, configurable depending on the required capabilities. The **executor** agent is supposed to actually execute the plan, possibly working "in the field", i.e., embedded in a mobile robot or some other ad-hoc facility or mechanism. Constraints can be used to codify knowledge about the environment, like obstacles, target coordinates, resources, depending on the problem domain.

Thus, given the input set $T_{PlanMax}, T_{Max}, G, N, C$, where G is the goal, N is the instance size of the problem to be solved (if applicable), C is the constraints set which models the dynamics and knowledge about the environment, the MAS operates via the following steps, not necessarily in sequence, but in parallel whenever it is possible:

(i) Decompose the overall goal into suitable subgoal;

(ii) For each subgoal, generate an a sub-plan within the $T_{PlanMax}$ deadline;

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

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

Since each ASP module may possibly find more than one plan for given (sub-)goal, it is useful (as said before) to apply a given metrics by which a plan could be preferred to another one. The proposed DALI ASP-MAS architecture is shown in Figure 1 and the agent behaviors are here described .

- **COORDINATOR** agent: this agent synchronizes all the actions of the MAS and updates the global state of goal solving. Its task are the following.
  (a) Ensure the proper activation of the MAS and overall self checking.
  (b) Interact with the external world and whenever needed acquire new constraints for the MAS or revise the present goals.
  (c) Control the $T_{PlanMax}$ and T_{Max} deadlines.
  (d) Decompose the goal into subgoals.
  (e) For each subgoal, instantiate a **META-PLANNER** agent, possibly providing as input the preference criterion for plan selection.
  (f) receive from each **META-PLANNER** agent the sub-plan to be executed up to $T_{PlanMax}$ and deploy the overall plan to the **EXECUTOR** agents set, each is in charge of sub-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** with new constraints to start the search for a new plan.
  (b) Generate input set of constraints and specific data for the **PLANNER** agent while monitoring its performances. If **PLANNER** agent does not deliver before $T_{PlanMax} - T_{Pt}$, 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. It also exploits the given preference criterium in order to select the plan which is closer to present preferences whenever the **PLANNER** returns more than one answer.
  (d) If requested by **COORDINATOR**, ask **PLANNER** for re-planning with updated input set of contraints.
- **PLANNER** agent, which receives as input the time constraints $T_{PlanMax}, T_{Max}, C_{\%}, N, F$ from **META-PLANNER** generate the ASP program which then generates all possible sub-plan via the 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). The $C_{\%}$ parameter encode knowledge about the sub-optimality of the desired plan type, which coincide with the Hamiltonian plan at 100%, or refers to sub-optimal plans for lower percentages.
- **EXECUTOR**: each agent puts into action in the real world the specific sub-plan provided by the **COORDINATOR**, if possible within the T_{Max} deadline, and notifies the **COORDINATOR** upon completion. The executor agent in general executes plans (also) embodied in a physical components in a Cyber-Physical System, and/or by means of robotic elements of various kinds. In Figure 1, **EXECUTOR** is designated as "field controller" as plan execution is situated into some environment.

Summarizing, the final execution made the EXECUTORs depends on the following information:

- timing parameters, ASP program templates, static constraints imposed by the designer
- selected goals and preferences by the user
- the environment model built upon sensors perceptions which define dynamic constraints
- consistency and self-checking rules in the knowledge base
- available energy and resources, which may also have non trivial impact on hardening the constraints set.

Since in general this is a hard-NP problem, most probably only sub-optimal plans can be generated, but with a controllable desirable quality by balancing user preferences, accuracy, and weak vs. hard constraints. The resulting behavior should be similar to what a rational human expert would do in similar circumstances, with the advantage of not being limited also by human errors due to over fatigue and less concentration. So the human could dedicate himself to supervise the overall system behavior under less cognitive load stress and intervene with appropriate common sense reasoning when needed, most probably when the system is producing too many trivial plans.

Abstract Case Study

The ASP-MAS architecture presented above has been inspired and motivated by a case-study that has been actually implemented and experimented, and presented in (Costantini, De Gasperis, and Nazzicone 2015). The overall goal in the case study is to explore an unknown territory upon occurrence of some kind of catastrophic-like disruptive event (earthquake, fire, flooding, terrorist attack, etc.). The similarity comes from the idea that after such event, most of the available geo-localized information can became obsolete in a very short time and important decision have to be made in order to save lives and/or deliver rescue services. So there is a contemporary need to re-scan the territory to know were is possible to engage rescue equipments, and to generate an actually rescue plan that covers the maximum possible area were is needed. So there are places were is impossible to go (i.e. *forbidden cells*) and places were victims have to be rescued (i.e. *to_reach cells*).

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, and their respective constraints set, can change in time as the scenario evolves.

For the sake of experiments, the EXECUTOR agent is embodied by a robot explorer/rescuerer [1] that each agent employs for exploration of the territory; this robot 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 may results computationally intractable with reasonable instance size (already from sizes ≥ 8, or 10 using the most recent ASP more efficient solvers), we resorted to sub-optimal solutions that the MAS is capable to generate, 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 (Calimeri and Zhou 2014) 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 (Leone et al. 2006a), 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 constraint, so as not to be forced to finding a Hamiltonian path.

```
reached(X,Y) :- move(1,1,X,Y).
reached(X2,Y2) :-
    reached(X1,Y1), move(X1,Y1,X2,Y2).
:~ 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 (Leone et al. 2006a) guarantees the desired coverage.

```
coverage(95).
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 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.

[1] not necessary a robot, also a human guided ambulance, or a combination of UAV and human guided vehicles

- The COORDINATOR agent partitions the territory that must be explored into a number of (possibly overlapping) sections (chessboards) of reasonable size (maximum 10x10 cells), each one to be assigned to a META-PLANNER instance.
- Each plan to be executed (exploration to be performed) is assigned to a separate (EXECUTOR)EXLORER agent, specifically assigned to that territory section. Each instance of the META-PLANNER agent relies upon its own associated instance of the planner agent.
- different preference policies can possibly be associated with different sections of the territory to be explored, according to directions provided by the user/environment.
- The COORDINATOR will devise re-planning for each portion of the territory for which the unreachable location have changed.

Reasonable metrics measure plans returned by the ASP module 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. 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.

Concluding Remarks

We have proposed an ASP-MAS architecture for flexible goal decomposition, plan formation and execution that delivers acceptable solution to complex problems under the "*bounded rationality principle*". In real application, a MAS for each (class of) goal(s) would be designed, implemented and located into the DALI cloud. In fact, all components of the MAS will be programmed according to the goal to be reached, i.e., to the problem to be solved. Each agent that needs to solve a goal refers to the suitable MAS. As mentioned, 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.

References

Aielli, F.; Ancona, D.; Caianiello, P.; Costantini, S.; De Gasperis, G.; Di Marco, A.; Ferrando, A.; and Mascardi, V. 2016. 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*, 15–26. Springer.

Baral, C. 2003. *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press.

Bevar, V.; Muccini, H.; Costantini, S.; De Gasperis, G.; and Tocchio, A. 2012. 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, 47–55. Springer, Berlin Heidelberg. Paper and demo.

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

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

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

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

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

Costantini, S., and Tocchio, A. 2002. A logic programming language for multi-agent systems. In *Logics in Artificial Intelligence, Proceedings of the 8th Europ. Conf.,JELIA 2002*, LNAI 2424. Springer-Verlag, Berlin.

Costantini, S., and Tocchio, A. 2004. The DALI logic programming agent-oriented language. In *Logics in Artificial Intelligence, Proceedings of the 9th European Conference, Jelia 2004*, LNAI 3229. Springer-Verlag, Berlin.

Costantini, S.; De Gasperis, G.; Pitoni, V.; and Salutari, A. 2017. 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*, 286–300. CEUR-WS.org. http://ceur-ws.org/Vol-1949/CILCpaper05.pdf.

Costantini, S.; De Gasperis, G.; and Nazzicone, G. 2015. 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.; and 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*, 285–292. Springer.

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

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

Costantini, S. 2011. Answer set modules for logical agents.

In de Moor, O.; Gottlob, G.; Furche, T.; and Sellers, A., eds., *Datalog Reloaded: First International Workshop, Datalog 2010*, volume 6702 of *LNCS*. Springer. Revised selected papers.

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

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

Dastani, M.; van Riemsdijk, B.; Dignum, F.; and Meyer, J.-J. C. 2004. A programming language for cognitive agents goal directed 3apl. In Dastani, M.; Dix, J.; and 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*, 111–130. Springer.

Dastani, M.; van Birna Riemsdijk, M.; and Meyer, J.-J. C. 2005. Programming multi-agent systems in 3apl. In *Multi-agent programming*. Springer. 39–67.

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

Elkabani, I.; Pontelli, E.; and Son, T. C. 2004. Smodels with clp and its applications: A simple and effective approach to aggregates in asp. In *International Conference on Logic Programming*, 73–89. Springer.

Fichera, L.; Messina, F.; Pappalardo, G.; and Santoro, C. 2017. A python framework for programming autonomous robots using a declarative approach. *Science of Computer Programming* 139:36–55.

Foundation for Intelligent Physical Agents. 2003. FIPA Interaction Protocolo Specifications.

Gebser, M.; Kaufmann, B.; Neumann, A.; and Schaub, T. 2007. clasp: A conflict-driven answer set solver. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, 260–265. Springer.

Gebser, M.; Maratea, M.; and Ricca, F. 2016. What's hot in the answer set programming competition. In *AAAI*, volume 16, 4327–4329.

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

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

Gigerenzer, G. 2004. Fast and frugal heuristics: The tools of bounded rationality. *Blackwell handbook of judgment and decision making* 62:88.

Gomez-Sanz, J. J., and Fuentes-Fernández, R. 2015. Understanding agent-oriented software engineering methodologies. *The Knowledge Engineering Review* 30(4):375–393.

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

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

Jiming, L. 2001. *Autonomous agents and multi-agent systems: explorations in learning, self-organization and adaptive computation*. World Scientific.

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

Knudson, M., and Tumer, K. 2011. Adaptive navigation for autonomous robots. *Robotics and Autonomous Systems* 59(6):410–420.

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

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

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

Lierler, Y. 2005. cmodels–sat-based disjunctive answer set solver. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, 447–451. Springer.

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

Romero, J.; Schaub, T.; and Son, T. C. 2017. Generalized answer set planning with incomplete information. In Bogaerts, B., and 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.

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

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