Dynamic Goal Decomposition and Planning in MAS for Highly Changing Environments

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

Abstract

This paper treats the problem of dynamic goal decomposition and planning in scenarios characterized by a strong inter-dependency between action and context, for instance the ones related to the rescue intervention in a territory upon occurrence of some kind of catastrophic event We propose an 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 allow for 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 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.

Introduction and Motivation

Scenarios characterized by **inter-dependency** between action and context are [17] 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 In such cases, data and knowledge about the territory is often outdated in a few seconds.

What is required there 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);

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

- **guide** 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).

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 [26; 5] 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 [11], 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 proposed in particular to adopt Answer Set Programming (ASP) modules, where ASP (cf., among many, [7; 3; 33; 38] 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 [39]. We in fact augmented the DALI agent-oriented language, invented and developed in our research group [25; 13; 12; 25], with a plugin for the invocation of answer set solvers. ASP modules can be exploited in agents in a variety of ways, 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 [8] 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 [4]. 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. In the first two sections 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 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 [27; 28] 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 [7; 3; 33; 38]. Planning is among the more suitable an successful applications of ASP: c.f., e.g., [37; 35] and the references therein, treating planning in ASP even under incomplete information. Several well-developed answer set solvers [39; 7] 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 [7].

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 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 represent 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 challenging sample applications proposed at the yearly ASP competition (cf. [9] for a recent report).

The DALI language: Framework and Applications

DALI [13] (cf. [12] for a comprehensive list of references) is an Agent-Oriented Logic Programming language. 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 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 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 what make 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 ctiticality for the agent.

The DALI communication architecture implements the DALI/FIPA protocol, which consists of the main FIPA primitives, plus few new primitives which are particular to DALI. The architecture also includes a filter on communication based on ontologies and forms of commonsense reasoning.

The DALI programming environment at current stage of development [25] 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 [16; 18]. 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 [16], 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 [18] 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 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 [1] 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 [31], 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 [14]).

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 [34; 6], GOAL [30; 29], 3APL [23; 24]) 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 [20; 19; 21].

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

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 are into slightly overlapping sub-territories. Other algorithms can be adopted to generate sub-plans, such as [36], [2] and [22]. The planner agent with the 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 1 and the agent behaviors are the following.

- **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.
  (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 the goal into subgoals.

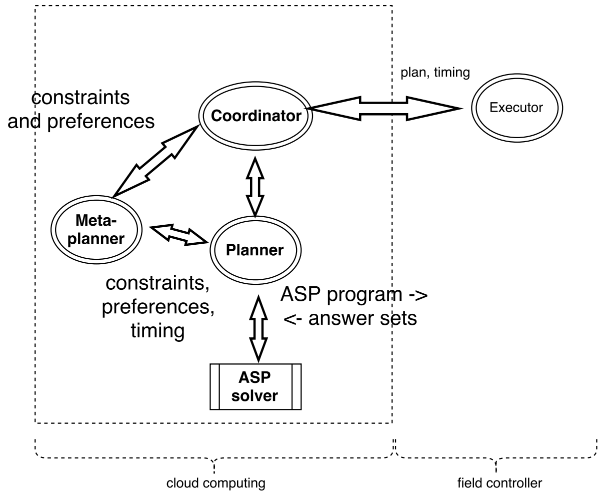

Figure 2: DALI MAS architecture: Coordinator, Meta-Planner, Planner, Explorer agents. The MAS can be deployed over a cloud computing architecture, thus distributing and balancing the required computational resources. The ASP module is executed via an external solver, configurable depending on the required 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.

  (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) (b) 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.
  (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 **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.
  (c) If requested by **COORDINATOR**4ent, ask **PLANNER** for re-planning with updated input.
- **PLANNER** agent, which receives as input the time constraints $T_{PlanMax}, T_{Max}$ and spatial constraints

$C_{\%}, N, F$ (ex: coverage, number of reach/do not reach cells) from **META-PLANNER** and generates possible plan 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.

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 [15]. 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 [10] 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 [32], 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 [32] 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 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 plan to be executed (exploration to be performed) is assigned to a separate EXPLORER agent, specifically assigned to that territory section. Each instance of the META-PLANNER agent relies upon its own associated copy 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 a MAS architecture for flexible goal decomposition, plan formation and execution [1]. 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/formalism. So, the proposed solution is not DALI-specific but rather can be generally adopted.

Acknowledgement

This research has been carried out thanks to internal funding from our department at University of L'Aquila. We thanks reviewers for helping us to improve the quality of this work.

References

[1] Federica Aielli, Davide Ancona, Pasquale Caianiello, Stefania Costantini, Giovanni De Gasperis, Antinisca Di Marco, Angelo Ferrando, and Viviana Mascardi. 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*, pages 15–26. Springer, 2016.

[2] Ron Alford, Vikas Shivashankar, Mark Roberts, Jeremy Frank, and David W Aha. Hierarchical planning: Relating task and goal decomposition with task sharing. In *IJCAI*, pages 3022–3029, 2016.

[3] Chitta Baral. *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press, 2003.

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

[5] Rafael H. Bordini, Lars Braubach, Mehdi Dastani, Amal El Fallah-Seghrouchni, Jorge J. Gómez-Sanz, João Leite, Gregory M. P. O'Hare, Alexander Pokahr, and Alessandro Ricci. A survey of programming languages and platforms for multi-agent systems. *Informatica (Slovenia)*, 30(1):33–44, 2006.

[6] Rafael H. Bordini and Jomi Fred Hübner. Semantics for the jason variant of agentspeak (plan failure and some internal actions). In Helder Coelho, Rudi Studer, and Michael Wooldridge, editors, *ECAI 2010 - 19th European Conference on Artificial Intelligence, Proceedings*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 635–640. IOS Press, 2010.

[7] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski (eds.). Answer set programming: Special issue. *AI Magazine*, 37(3), 2016.

[8] Maurice Bruynooghe, Theofrastos Mantadelis, Angelika Kimmig, Bernd Gutmann, Joost Vennekens, Gerda Janssens, and Luc De Raedt. 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*, pages 719–724, 2010.

[9] Francesco Calimeri, Giovambattista Ianni, Thomas Krennwallner, and Francesco Ricca. The answer set programming competition. *AI Magazine*, 33(4):114–118, 2012.

[10] Francesco Calimeri and Neng-Fa Zhou. 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`.

[11] S. Costantini. Answer set modules for logical agents. In Oege de Moor, Georg Gottlob, Tim Furche, and Andrew Sellers, editors, *Datalog Reloaded: First International Workshop, Datalog 2010*, volume 6702 of *LNCS*. Springer, 2011. Revised selected papers.

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

[13] S. Costantini and A. Tocchio. 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, 2004.

[14] Stefania Costantini. Ace: a flexible environment for complex event processing in logical agents. In Luciano Baresi Matteo Baldoni and Mehdi Dastani, editors, *Engineering Multi-Agent Systems, Third International Workshop, EMAS 2015, Revised Selected Papers*, volume 9318 of *Lecture Notes in Computer Science*. Springer, 2015.

[15] Stefania Costantini, Giovanni De Gasperis, and Giulio Nazzicone. Exploration of unknown territory via DALI agents and ASP modules. In Sigeru Omatu, Qutaibah M. Malluhi, Sara Rodríguez-González, Grzegorz Bocewicz, Edgardo Bucciarelli, Gianfranco Giulioni, and Farkhund Iqba, editors, *Distributed Computing and Artificial Intelligence, 12th International*

[1]A previous version of this paper has 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.

Conference, DCAI 2015, Salamanca, Spain, June 3-5, 2015, volume 373 of *Advances in Intelligent Systems and Computing*, pages 285–292. Springer, 2015.

[16] Stefania Costantini, Giovanni De Gasperis, and Giulio Nazzicone. DALI for cognitive robotics: Principles and prototype implementation. In Yuliya Lierler and Walid Taha, editors, *Practical Aspects of Declarative Languages - 19th International Symposium, Proceedings*, volume 10137 of *Lecture Notes in Computer Science*, pages 152–162. Springer, 2017.

[17] Stefania Costantini, Giovanni De Gasperis, Giulio Nazzicone, and Laura Tarantino. Context and action: A unitary vision within a logic-based multi-agent environment. *Lecture Notes in Information Systems and Organisation*, 18:97–111, 2016.

[18] Stefania Costantini, Giovanni De Gasperis, Valentina Pitoni, and Agnese Salutari. 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*, pages 286–300. CEUR-WS.org, 2017. http://ceur-ws.org/Vol-1949/CILCpaper05.pdf.

[19] Stefania Costantini and Andrea Formisano. Modeling preferences and conditional preferences on resource consumption and production in ASP. *Journal of of Algorithms in Cognition, Informatics and Logic*, 64(1), 2009.

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

[21] Stefania Costantini, Andrea Formisano, and Davide Petturiti. Extending and implementing RASP. *Fundam. Inform.*, 105(1-2):1–33, 2010.

[22] Michael T Cox, Dustin Dannenhauer, and Sravya Kondrakunta. Goal operations for cognitive systems. In *AAAI*, pages 4385–4391, 2017.

[23] Mehdi Dastani, Birna van Riemsdijk, Frank Dignum, and John-Jules Ch. Meyer. A programming language for cognitive agents goal directed 3apl. In Mehdi Dastani, Jürgen Dix, and Amal El Fallah-Seghrouchni, editors, *Programming Multi-Agent Systems, First International Workshop, PROMAS 2003, Selected Revised and Invited Papers*, volume 3067 of *Lecture Notes in Computer Science*, pages 111–130. Springer, 2004.

[24] Mehdi Dastani, M. Birna van Riemsdijk, and John-Jules Ch. Meyer. Programming multi-agent systems in 3APL. In Rafael H. Bordini, editor, *Multi-Agent Programming: Languages, Platforms and Applications*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, pages 39–67. Springer, 2005.

[25] Giovanni De Gasperis, Stefania Costantini, and Giulio Nazzicone. Dali multi agent systems framework, doi 10.5281/zenodo.11042. DALI GitHub Software Repository, July 2014. DALI: `http://github.com/AAAI-DISIM-UnivAQ/DALI`.

[26] Michael Fisher, Rafael H. Bordini, Benjamin Hirsch, and Paolo Torroni. Computational logics and agents: a road map of current technologies and future trends. *Computational Intelligence Journal*, 23(1):61–91, 2007.

[27] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert Kowalski and Kenneth Bowen, editors, *Proceedings of the 5th International Conference and Symposium on Logic Programming (ICLP/SLP'88)*, pages 1070–1080. The MIT Press, 1988.

[28] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[29] Koen Hindriks. A verification logic for goal agents. In Mehdi M. Dastani, Koen Hindriks, and John-Jules Charles Meyer, editors, *Specification and Verification of Multi-agent Systems*. Springer, 2010.

[30] Koen V. Hindriks. Programming rational agents in goal. In *Multi-Agent Programming*, pages 119–157. Springer US, 2009.

[31] Siddhartha Kumar Khaitan and James D. McCalley. Design techniques and applications of cyberphysical systems: A survey. *IEEE Systems Journal*, 9(2):350–365, 2015.

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

[33] Nicola Leone. Logic programming and nonmonotonic reasoning: From theory to systems and applications. In Chitta Baral, Gerhard Brewka, and John Schlipf, editors, *Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007*, 2007.

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

[35] Javier Romero, Torsten Schaub, and Tran Cao Son. Generalized answer set planning with incomplete information. In Bart Bogaerts and Amelia Harrison, editors, *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.

[36] Vikas Shivashankar, Ronald Alford, Ugur Kuter, and Dana S Nau. The godel planning system: A more perfect union of domain-independent and hierarchical planning. In *IJCAI*, pages 2380–2386, 2013.

[37] Tran Cao Son. Answer set programming and its applications in planning and multi-agent systems. In Marcello Balduccini and Tomi Janhunen, editors, *Logic Programming and Nonmonotonic Reasoning - 14th International Conference, LPNMR 2017, Proceedings*, volume 10377 of *Lecture Notes in Computer Science*, pages 23–35. Springer, 2017.

[38] Miroslaw Truszczyński. Logic programming for knowledge representation. In Verónica Dahl and Ilkka Niemelä, editors, *Logic Programming, 23rd International Conference, ICLP 2007*, pages 76–88, 2007.

[39] 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`.