DALI: A multi agent system framework for the web, cognitive robotic and complex event processing

Stefania Costantini, Giovanni De Gasperis, Valentina Pitoni, and Agnese Salutari

DISIM, Università di L'Aquila, Italy

Abstract. This paper illustrates advances over existing implementation work performed on DALI, which is a logic prolog-based framework for defining agents and Multi-Agent Systems (MAS) developed at University of L'Aquila since 1999, and publicly available. In previous work, DALI features were already extended in view of cognitive robotic applications and Complex Event Processing. Here, we show how to define and implement more efficient and versatile Multi-Agent Systems, capable to interact with real robots.

Keywords: multi agent systems, cognitive robotics, complex event processing, logic programming

1 Introduction

Quoting from `http://www.ieee-ras.org/cognitive-robotics`,

> There is growing need for robots that can interact safely with people in everyday situations. These robots have to be able to anticipate the effects of their own actions as well as the actions and needs of the people around them. To achieve this, two streams of research need to merge, one concerned with physical systems specifically designed to interact with unconstrained environments and another focusing on control architectures that explicitly take into account the need to acquire and use experience. The merging of these two areas has brought about the field of Cognitive Robotics. This is a multi-disciplinary science that draws on research in adaptive robotics as well as cognitive science and artificial intelligence, and often exploits models based on biological cognition. Cognitive robots achieve their goals by perceiving their environment, paying attention to the events that matter, planning what to do, anticipating the outcome of their actions and the actions of other agents, and learning from the resultant interaction. They deal with the inherent uncertainty of natural environments by continually learning, reasoning, and sharing their knowledge.

Several papers on cognitive robotics can be found in the proceedings of main Conferences on Artificial Intelligence (e.g., ECAI, IJCAI) and on Agents (e.g., AAMAS). Cognitive robotic systems able to interact safely with people in everyday situations can potentially help in any situation where there are persons in need of special assistance, primarily the elderly and the disabled, but possibly also children (not as a substitute but as a support to parents, family and caregivers). With the increase of life expectancy and the decrease of resources available to public health systems, the importance of developing "intelligent" adaptive robots can be particularly appreciated in view of the societal

issue of providing effective support to many people in need while making the use of helth-related resources more effective and rational. These robots have to be able to anticipate the effects of their own actions as well as the actions and needs of the people around them.

Cognitive Robotics is thus potentially able to alleviate the healthcare and social security systems from the burden of having to provide full-time highly specialized human assistants while allowing elderly people to leave at home rather than be moved to an institution. In [1] we have outlined a comprehensive system called Friendly&Kind, for short F&K based upon a reasoning "core" which is able to integrate information gathered from a variety of knowledge sources, and devise a good (if not optimal) course of action in view of each patient's welfare and of the overall system's possibilities; logical agents are envisaged for monitoring patients, where a patient's personal agent can be embodied in a robot. Some of the monitoring can be performed locally, while some other will be deferred to the F&K reasoning systems or to the human operators.

Understandably, as demonstrated by experiments performed, e.g., by the group of Prof. Johan Hoorn at Vrije Universiteit Amsterdam about "social robotics", humans prefer friendly interfaces and robots that show some kind of intelligent and also affective and "emotional" behavior. Such experiments consider not only Artificial Intelligence aspects, but also the impact of robots on the user. There is interesting ongoing work also from the point of view of ethics [29], interaction with the disabled [28] and even acceptable robot appearance [27]. Some of this work is reported in a famous documentary "Alice cares" (`https://vimeo.com/116760085`) which shows the positive interaction among three old women and a friendly humanoid care robot (a scene from the documentary is reported in Figure1). From a cognitive point of view and for exploring the social acceptability of robots as human companions these experiments are certainly of great importance. However, as concerns really "intelligent" behavior the robots used in the experiments are still (at least partly) under remote control of a human operator. In the perspective of making such robots really intelligent and autonomous, research results from many fields of Artificial Intelligence, Automated Reasoning and Intelligent Software Agents can be usefully exploited.

Fig. 1.

We strongly believe that in this and other fields it can be advantageous to define a robot's cognitive part as a logical agent or Multi-Agent System defined via declarative

agent-oriented languages. There are many logic agent-oriented languages and architectures in computational logic apt to these aim, among which MetateM, 3APL, GOAL, AgentSpeak, Impact, KGP and DALI, that might be usefully exploited in robotics (or have potential robotic applications).

However, the DALI language in particular [18, 19] has been empowered and experimented over the years concerning capabilities for the definition and management of an agent's memory and experience and for user monitoring and training also by learning new behavioral patterns; DALI agents are able to perform complex event processing, and to dynamically check and modify their own behavior also in terms of a special interval temporal logic (cf. [9, 13, 10, 7] and the references therein). In [12] we have discussed an extension to the basic DALI implementation that allows action commands to be exchanged between DALI agents and any robotic platform by using the YARP middleware. A short practical case study showed how a DALI MAS can control the iCub open-source robot simulator by exchanging asynchronous JSON events over the new multi-standard DALI network bus. In addition, we implemented ServerDALI which allows to allocate DALI agents and MAS on a server, so as to be accessible from an external environment, also via web or mobile applications. This is relevant, as for instance in the architecture of Figure2 the caregiver agents will presumably be copies of the same one, to which robots' cognitive functioning can refer; so, a cloud solution eliminates the need of equipping the (possibly diverse) robot hardware with sophisticated software; moreover, computationally heavy automated reasoning tasks can be more efficiently executed on the server.

The contribution of [12] was twofold: on the one hand, we re-elaborated and extended past work on DALI in the perspective of robotic applications for the care of persons in need; on the other hand, we realized and experimented a practical implementation constructed out of open-source components, with the aim to employ DALI agents as a robot's "brain"; we were working however over robot simulators rather than real robot hardware.

This paper illustrates advances over the pre-existing implementation work performed with the aim to obtain more efficient and versatile Multi-Agent Systems, capable to interact with real robots (rather than with simulators). In particular, the new implementation adds the following non-trivial features: (i) heterogeneity, i.e., a MAS can now include not only DALI agents but also agents written in other logic-based-language; (ii) run-time reconfiguration of a MAS, i.e., agents can be started, stopped, suspended during the MAS operation; (iii) real potential parallelism, i.e., agents' execution is asynchronous (differently from previous implementation) and thus much more versatile and efficient; (iv) via the Redis open-source database/communication infrastructure [1], agents can seemingly interact with existing robot-oriented technologies and can thus command robots. Notice that in this paper we are not concerned with physical aspects concerning sensors, actuators, vision, etc., that are widely studied by specialists. We model hardware subcomponents via a layer of abstraction that builds a network of sources and sinks of asynchronous events (or commands) with associated values.

The paper is organized as follows. In Section 2 we recall the basic DALI language; in Section 3 we illustrate, by means of small though significant examples, the potential

[1] `http://redis.io`, last accessed June 2017

applicability of DALI in robotic user monitoring and training. In Section 4 we recall the extension to robotics of the DALI implementation, and in Section 5 we intruduce the new further extensions that we have recently developed. Finally, in Section 6 we conclude.

2 The basic DALI language and architecture

DALI [18, 19] is an Agent-Oriented Logic Programming language that is publicly available on GitHub [11]. DALI agents are able to deal with several kinds of events: external events, internal, present and past events.

External events are syntactically indicated by the postfix *E* and we have reactive rules like $ev_E :> Reaction$: if an agent perceives the event *ev* then she will react as described by *Reaction*. 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 postfix *N*.

Actions (indicated with postfix *A*) may have or not preconditions: in the former case, the actions are defined by actions rules like $act_A :< Precondition$, in the latter case they are just action atoms. Similarly to events, actions that already took place are recorded as past actions.

Internal events is the feature that makes DALI agent agents proactive. An internal event is syntactically indicated by the postfix *I*, and its description is composed of two rules. The first one, $ev : -Precondition$, contains the conditions (knowledge, past events, procedures, etc.) that must be true so that the reaction (in the second rule $ev_I :> ActionToDo_A$) may happen. Internal events are automatically attempted with a default frequency customizable by means of directives in the initialization file.

The DALI communication architecture [20] implements the DALI/FIPA protocol, which consists of the main FIPA primitives, plus few new primitives which are particular to DALI and provides the possibility of defining meta-rules for filtering incoming and out-coming messages, and for using external ontologies.

DALI provides a plugin to an answer set solver, so complex reasoning tasks such as, e.g., planning and preference handling can be performed in Answer Set Programming (ASP), which is a state-of-the art technology for dealing with hard computational problems (cf., among many, [3] and the references therein).

"*Complex Event Processing*" (CEP) has evolved into the paradigm of choice for the development of monitoring vast quantities of event data to make automated decisions and take time-critical actions. This is particularly important in software agents, in fact, agents and multi-agent systems are able to manage rapid change and thus to allow for scalability in applications aimed at supporting the ever-increasing level of interaction. Work on CEP in DALI is presented in [15] and in [5], which discuss the issue of selecting different reactive patterns by means of preferences. Such preferences can be also defined in terms of "possible worlds" elicited from a declarative description of a current or hypothetical situation, and can depend upon past events, and the specific sequence in which they occurred. [17] introduces *Event-Action modules* that allow an agent to: aggregate simple events into complex ones, also according to constraints; check events

that occur w.r.t. expectations; cope with events possibly occurring contextually to certain other events; detect anomalies; decide actions to be performed in both normal or anomalous cases, according to a number of issues among which we may include context, role, circumstances, past experience, etc.

3 DALI advanced features and possible applications to robotics

The robotic applications that we envisage concern, as outlined above, user monitoring and training in any context, but especially for the care of elderly and disabled persons.

Since [14] in fact, in our setting (see Figure 2) agents interact with users (i) with the objective of training them in some particular task, and (ii) with the aim of monitoring them for ensuring some degree of consistence and coherence in user behavior.

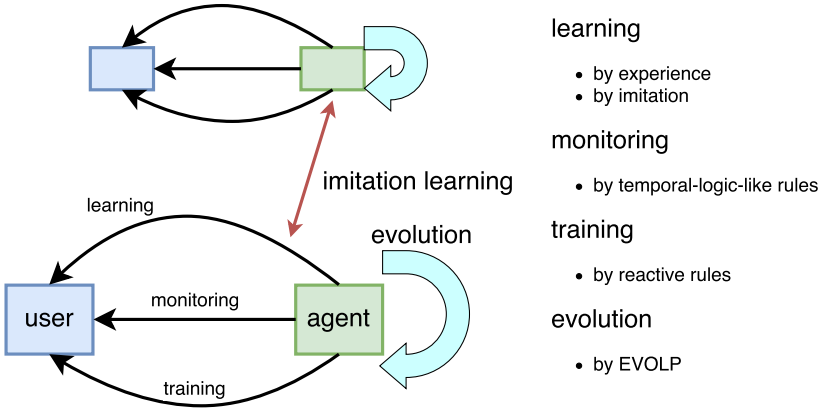

Fig. 2. Agent interaction model

Agents are able to be aware by prior knowledge or via some form of learning of the behavioral patterns that the user is adopting, and to learn rules and plans also from other agents (by imitation or being told); notice that a logical account of the program modifications which are necessary in a learning process is provided by the EVOLP approach [2, 24], as shown in the Figure 2. In the simple example below, an agent has been somehow able to learn that the user normally takes a drink when coming back home. This can be represented by a rule such as:

$$drink \text{ :- } arrive\_home.$$

This learned rule can possibly be associated with a certainty factor. When the rule becomes later confronted with subsequent experience, its certainty factor will be updated accordingly. Whenever this factor exceeds a threshold, this may lead to assert new meta-rules, such as:

$$USUALLY\ drink\ WHEN arrive\_home.$$

User monitoring can be performed via temporal-logic-like rules like the following one:

$$NEVER\ drink\_alchool\ AND\ take\_medicine.$$

Such a rule acts as a constraint which has priority over former ones; so, the agent will actively discourage the user to drink while taking medicines. In [6] the semantics of such expressions is defined, also in relation to the possibility of defining the *interval* where some events/actions must or must not occur.

The following example concerns a robot aiding to supervise a baby, thus relieving caregivers from some of their tasks. If the baby is hungry, then the robot should feed the baby with available baby food (feeding is an action, indicated with postfix A) paying attention to choose the healthier among those that the baby likes. Conjunction $food(F),\ available(F)$ provides a number of values for F, among which one is chosen. In particular, the choice will correspond to a maximum in the partial order imposed by the binary predicates $best\_preferred$ and $healthier$ in the given order. This construct for complex preference, the p-set, was originally introduced in [16].

$$baby\_is\_hungryE :>$$
$$\{feed\_babyA(F)\ :\ food(F),\ available\_babyf(F)\ :\ best\_preferred, healthier\}.$$

In the example below, the robot again assists parents taking care of a child. The child has to go to school (mandatory goal, indicated by postfix G) and is about to skip breakfast because she prefers cereals that unfortunately are finished. The agent, based upon the monitoring condition (never skip breakfast) will be able to suggest alternative food, in particular the best preferred among available options.

$$go\_to\_schoolG :\ NEVER\ skip\_breakfast(D) ::\ cereals\_finished :::$$
$$suggestA(alternative\_food)\ IN\ \{cookies, cake\_slice : best\_preferred\}.$$

The monitoring component can however also include meta-axioms such as for instance the following one, which states that a user action which is necessary to reach a mandatory objective should necessarily be undertaken. The agent can fulfill this statement either by convincing the user to do so, or to resort to human caregivers' help:

$$ALWAYS\ do(user, A)\ WHEN\ mandatory\_goal(G), required(G, A)$$

Such a meta-rule could be applied to practical cases such as the following:

$$mandatory\_goal(healthy).$$
$$required(healthy, take\_medicineA).$$

ASP modules can be exploited in order to plan actions which might be performed in given situations, and to extract *necessary* actions, which are those actions included in all possible plans. Given ASP module M (defined in a separate text file), in the example below reaction to event evE can be either any action which can be inferred (from M) as a possible reaction, or a *necessary* action, again according to M. Events are indicated with postfix E, reaction is indicated with $:>$. Connective $>$ expresses preference: the former option is preferred over the latter if the condition after the :- holds; $necessary$

and $action$ are distinguished predicates applicable over ASP modules' results. So, in this sample rule necessary actions are preferred in a critical situation. Otherwise, any of the two options may be taken.

$$evE :> necessary(M, N)|action(M, A) \ : \ M \ > \ A \text{ :- } critical\_situation.$$

The above examples are witnesses of a re-elaboration of past work on DALI in the perspective of cognitive robotics applications. Though small, the examples should have practically demonstrated that DALI has indeed the potential for acting as an agent language in this realm. However, a suitable interface between DALI agents and robotic hardware or simulators was lacking. Such an interface has been recently designed and implemented, and is presented in the next sections.

4 The extended DALI implementation

As discussed in [12], the DALI programming environment at the current stage of development [11] offers a multi-platform folder environment (for both Linux and Windows operative systems) including Sicstus Prolog programs (as DALI is implemented in Sicstus), shells scripts, and Python scripts.

For the development of DALI agents and MAS, a programmer can simply use any text editor to write DALI agent programs and the necessary start/configuration scripts; more proficiently, she could use a web-based system-independent integrated development environment where agents editing is managed through an HTML5/AJAX-based online editor, with start/stop command buttons and agents logs output for runtime verification, handling signals and events from the DALI engine running in the background (that isn't a actually an IDE, but we can use these features to develop a DALI MAS). The system is designed so as to be able to interact with external services by means of JSON data events. An external service can be for instance a virtual robotics simulator so that an entire complex anthropomorphic cognitive robot like the iCub [26] could be controlled by a DALI MAS.

The software components diagram in 3 shows how DALI has been encapsulated and integrated with other modules through a Python "glue code" layer, called PyDALI. Each DALI agent is an instance of the Prolog program "DALI Interpreter". The multi-platform open source library *pexpect* (`http://github.com/pexpect/pexpect`) has been adopted for building a Python middle layer to automate the interaction with the Sicstus Prolog environment, seen as an instance of the class PySicstus. In this way, by abstracting via the PyDALI class, a DALI agent instance process can be configured, loaded, started, executed and terminated. A MAS can then be handled via the most abstract class "MAS".

The Python code can then been imported into any Python program, thus allowing the interaction of DALI agents with other software modules/server/clients by means of asynchronous JSON events.

The *Multi-standard DALI Bus* is in essence a middle layer communication protocol that converts any JSON event coming from the outside world to an internal FIPA event

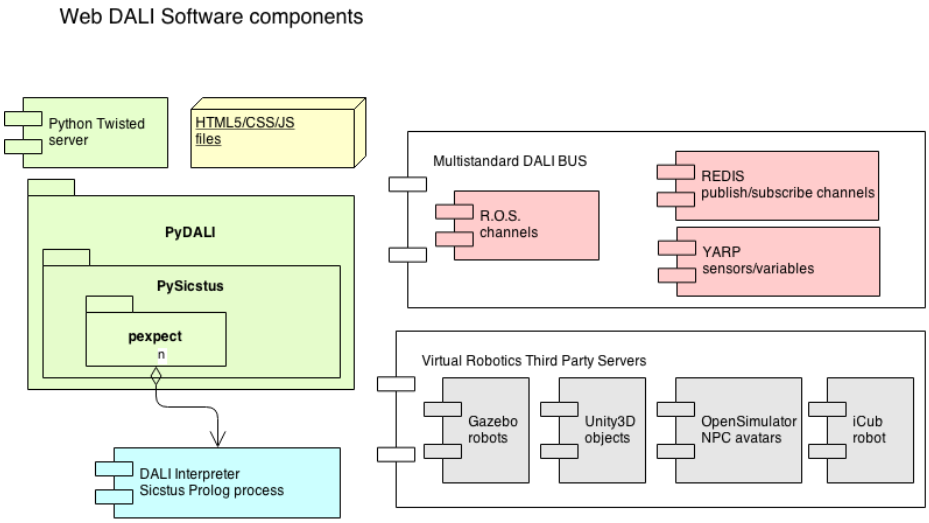

Fig. 3. Software components diagram of the extended DALI architecture

in a Linda tuple space [2], that the DALI MAS thus receives as an external event. Specific actions performed within the DALI MAS can vice versa generate FIPA events that are converted into JSON event so as to send commands to external actuators, that can be either real robotic actuators or virtual robotic components. A typical runtime deployment diagram is depicted in Figure 4.

The central Multi-standard DALI bus collects asynchronous data events from different sources, translating them into counterparts in the Linda tuple space whenever an agent is the destination. It also collects action messages from agents and translates them into JSON structures compatible with the destination. There may also be external sensors that directly generate Linda tuple messages, or external sensors mediated by the Python container.

5 Koiné DALI

Koiné DALI is the new extension of DALI that we present in this paper. It represents the evolution of ServerDALI [12], augmented with advanced capabilities obtained via the integration with the Redis open source "data structure server" which is available on many cloud computing service providers such as the Google Cloud Platform or the Amazon Elastic Cloud. Koiné DALI makes it possible to exchange data and communicate events to a DALI MAS in a very general way. Koiné DALI (Figure 5) offers the following functionalities:

[2] *Linda* is a model of coordination and communication among parallel processes providing a logically global *associative memory*, called a "tuplespace", in which processes store and retrieve tuples. It is available for Sicstus Prolog and it is therefore used as a communication middleware in the DALI implementation.

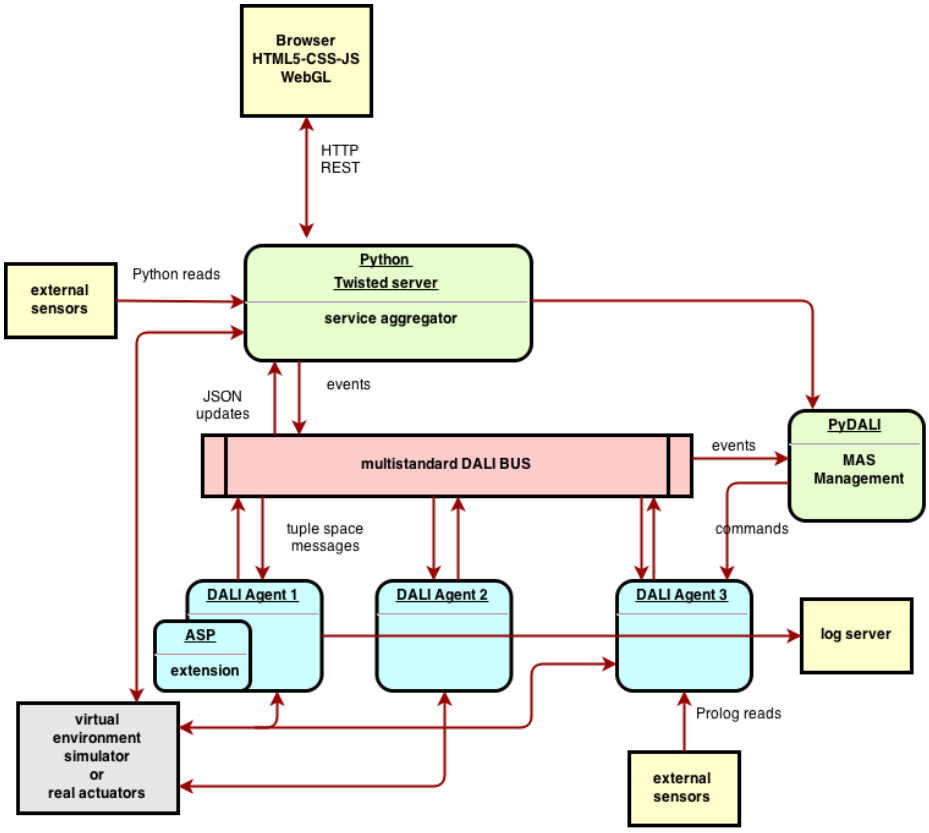

Fig. 4. Runtime deployment diagram of the extended DALI architecture

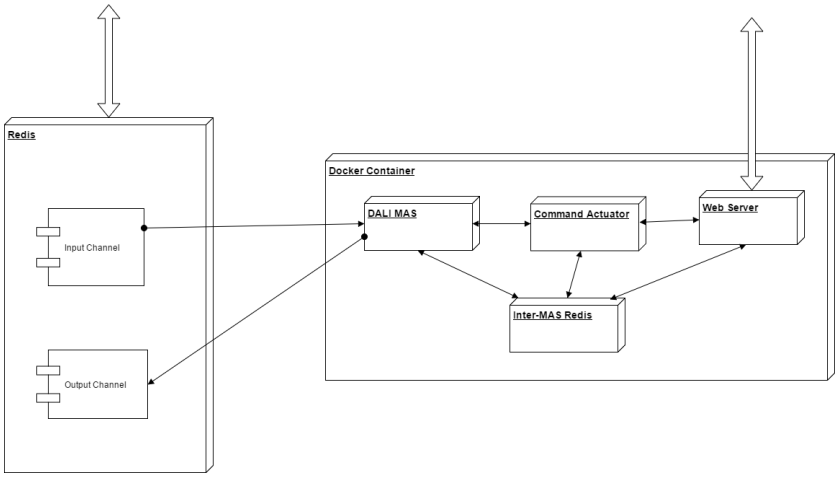

Fig. 5. Koiné DALI General Deployment Diagram

- A DALI MAS can be easily integrated with other applications using Redis both as a Database server and as a communication channel: events can be stored into Redis queues (FIFO) and then delivered to the MAS agents;
- the user can customize a MAS configuration (i.e., clone agents, stop agents, etc.) at runtime via a Web Server Interface: the Command Actuator program will implement the changes on the MAS and Inter-MAS Redis will be used as an internal communication channel;
- Agents composing a DALI MAS can use Inter-MAS Redis also as a database server to their own purposes;
- a Koiné DALI MAS can cooperate without problems with other MASs, programmed in other languages, and with object-oriented applications.

To create a Koiné DALI MAS, we need to add to a DALI MAS the followings:

- Redis2LINDA and StringESE libraries (`https://github.com/AAAI-DISIM-UnivAQ/Redis2LINDA-stringESE`) to deliver messages from Redis to MAS
- RedisClient library (`https://github.com/AAAI-DISIM-UnivAQ/RedisClient`) to deliver messages from MAS agents to Redis
- ServerPyProlog library (`https://github.com/AAAI-DISIM-UnivAQ/ServerPyProlog` to exchange data between Python programs (or servers) and Prolog programs or Prolog based agents, (if you need to use PHP instead of Python, see ServerProlog: `https://github.com/agnsal/ServerDALI/tree/master/serverPROLOG`)
- The special startmas startmasMary (`https://github.com/agnsal/ServerDALImas/blob/master/startmasMary.sh`) if you need to run the MAS in a Docker container

The MASA (MAS Administrator) (Figure 6) is an agent that supervises the MAS. When an event arrives to Redis Input Channel, the Event Proxy converts it into a standard format and sends it to MASA via the Linda channel. MASA sends it to the proper agent (depending on the configuration) after a second conversion into the format which is accepted by that particular agent; the agent will thus be able to update its KB (Knowledge Base), make inferences and send back its results to Redis Output channel. All internal communication mechanisms are based on the publisher/subscriber asynchronous design pattern, as it is customary in the software engineering community.

The new implementation also enables agents to clean their Knowledge Base from events that are older than a certain threshold (defined in the agent's configuration files).

We have successfully implemented and tested these functionalities, and we are now experimenting Koiné DALI ability to allow configuration changes to be made at runtime (the parts under test are distinguished by the violet arrows in Figure 6).

In the new implementation, Koiné DALI agents work asynchronously (Figure 7) after a handshake, and could therefore be executed in parallel over suitable hardware. The handshake is the first task that a Koiné DALI MAS performs and it is necessary for the bootstrap operation of the MAS (all the agents have to be operative). MASA delivers arriving events to proper agents and different agents can work at the same time on different events; the resulting MAS is therefore quite efficient. MASA is like

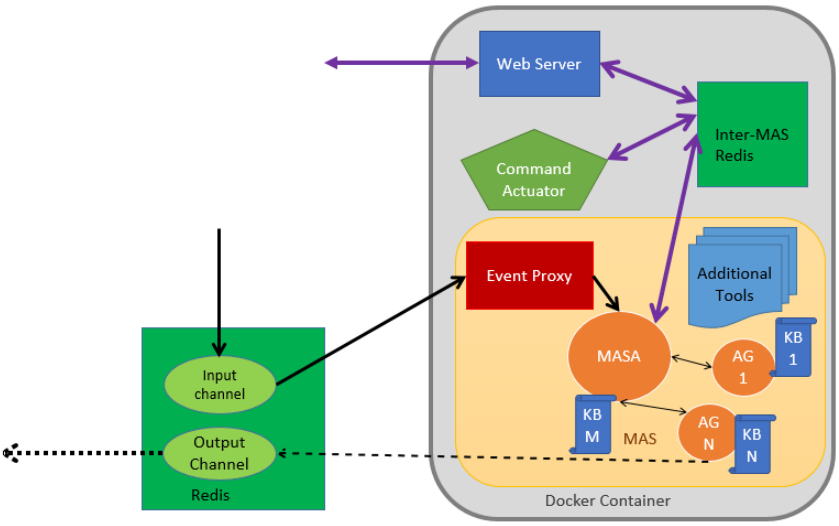

Fig. 6. Koiné DALI MAS in detail

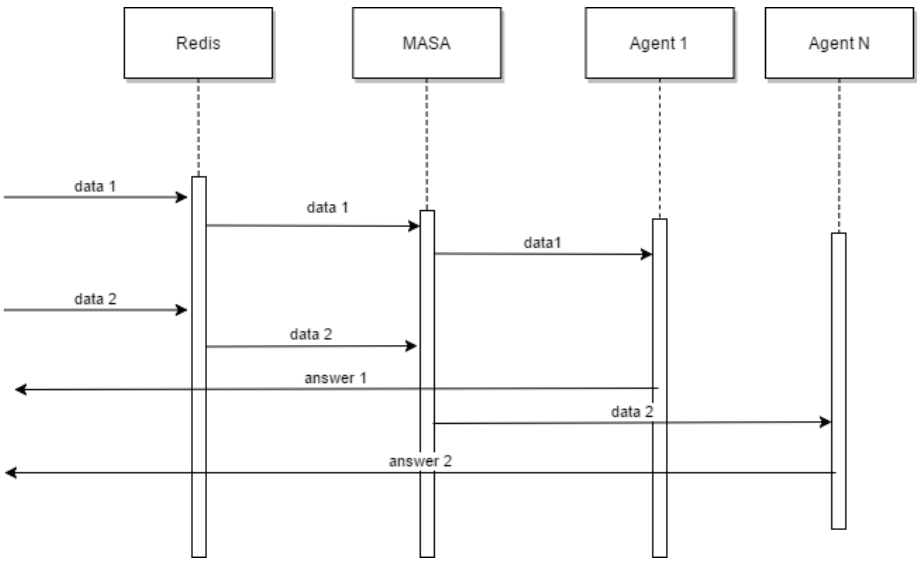

Fig. 7. Koiné DALI Sequence Diagram

the center of a "stellar system" where the number of the agents can vary according to specific needs.

Finally, we can see a little example of a MAS, composed of two agents, the MASA and a normal agent, that are able to communicate with Redis (Figure 9). At the beginning they perform the handshake, then the MASA waits for data/events (i.e. an hello-world event) from Redis and delivers them to the qualified agent (the normal agent, in this case). The competent agent reacts to the event by sending its result to Redis.

```
:- dynamic count/1.
:- assert(count(0)).
:- dynamic event/1.
helloE(X) :>
    messageA(X,send_message(hi,masa)),
    print(X), print(' said hello!'), nl,
    count(N),M is N+1,retract(count(_)),assert(count(M)).
ready :- numAgents(K), count(K).
readyI :>
    retract(count(_)),
    print('All the agents are ready!'), nl,
    print('MAS is waiting'), nl.
rediseE(X) :>
    print('Redis told me: '), print(X), nl, atom(X),
    pulisciStringa(X, Prolog), print(' corresponding to: '),
    print(Prolog), nl,
    notebook(Prolog).
notebook(L) :-
    open('notebook.pl', write, W),
    write(W,'event('),
    write(W, L),
    write(W,').'),
    close(W),
    compile('notebook.pl'),
    ( event(helloWorld) ->
        competent(Ag, helloWorld),
        print('I say to do helloWorld to competent agent: '),
        print(Ag), print('.'), nl,
        messageA(Ag,send_message(helloWorld, masa))
    ).
```

(a) Agent Type MASA

```
:- dynamic state/1.
:- assert(state(0)).

hello :- state(0).
helloI :>
    print('I say hello to MASA!'), nl,
    messageA(masa,send_message(hello(Me),Me)).
hiE :>
    retract(state(0)),
    assert(state(1)),
    print('MASA told hello back!'), nl.
helloWorldE :>
    print('HELLO WORLD!'), nl,
    print('I send helloWorld to Redis'), nl,
    mas_send('HELLO_WORLD_by_NormalAgent1').
```

(b) Agent Type Normal Agent

Fig. 8. Little example of a Koiné DALI MAS Prolog code (Koiné libraries inclusion has been omitted)

5.1 Redis, the communication interfaces for ROS, YARP and MQTT subsystems

The inclusion of the Redis channels into the software architecture as internal and external communication allowed us to build several technological plugins oriented to the application domain, in this case robotics. We started with ROS, YARP and MQTT plugins to cover a large range of hardware devices nowadays available on the market.

ROS (Robot Operating System) [3] "*provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS is licensed under an open source, BSD license*".

YARP, Yet An other Robotic Platform, [25] [23] "*supports building a robot control system as a collection of programs communicating in a peer-to-peer way, with an extensible family of connection types (tcp, udp, multicast, local, MPI, mjpg-over-http, XML/RPC, tcpros,. . .) that can be swapped in and out to match your needs.* ".

[3] `http://ros.org`, last accessed June 2017

MQTT is a machine-to-machine (M2M)/“Internet of Things” connectivity protocol. It was designed as an extremely lightweight publish/subscribe messaging transport. It is useful for connections with remote locations where a small code footprint is required and/or network bandwidth is at a premium.

```
import ServerPyProlog as SPP
import redis

translator=SPP.ServerPyProlog()
...
R1 = redis.Redis() # R1 is a Redis instance
pubsub1 = R1.pubsub() # Via pubsub1 we can listen on R1
pubsub1.subscribe('toMAS')
#We are listening to R1/toMAS channel
...
R2 = redis.Redis()
R2.publish('LINDAchannel', "helloWorld")
#We are publishing the str "helloWorld" on R2/toRedis channel
```

Fig. 9. Little example of a Phyton program communicating with a Koiné DALI MAS via Redis

As said before, a Koiné DALI MAS can communicate via Redis, sending and receiving messages (consisting of strings) through its ports. This mechanism is very similar to ROS, YARP and MQTT technologies, so we have been able to develop libraries that will make it possible to create a communication between Koiné DALI MASs and these three technologies and in perspective with physical robots.

The code snippet in Figure 10 copes with listening and publishing on Redis channels only; however, Redis can provide complex data structures as well, so for instance we could store strings in a FIFO queue and extract them from there as needed (from both the MAS and other technologically different components). We have been been able to deploy Redis compatibility libraries for ROS, YARP and MQTT, in order to make Koiné DALI MASs able to control robots.

This example of YARP in Python/DALI shows how similar its behavior and Redis' one are. The program, typically embedded onboard a robot, generates raw data and sends them to the YARP port “/sender”. This port could be connected to a “/receiver” YARP port by means of a channel configurator. Then, a Python program could register itself as the handler of the “/receiver” port, thus translating the data into a Linda tuple space, like shown in Figure 3.

6 Conclusions

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 [12] and in the present paper we have in fact 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 [12], the DALI framework has been

```
from time import sleep
from pydali import YARP_Agent, MAS

ag1 = YARP_Agent('one', '/receiver')
ag1.setSource('''
:- writeLog('agent ONE started').
receiverDataE(X)  :>
writeLog('received data: '), write(X), nl.
''')
myMAS = MAS('YARP_MAS')
myMAS.add(ag1)
myMAS.start(True, 'YARP test')
sleep(500)
myMAS.terminate()
```

Fig. 10. Python program that allocates a MAS made of just a single DALI agent, creates a connection to the YARP port "/receiver" and transparently translates data events into FIPA `inform` messages.

extended 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 DALI 2.0 framework [12] 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 [12] allows a DALI MAS to be integrated into any practical environment. In this paper we have illustrated the new "Koiné DALI" framework that we have recently developed and that we are successfully testing. As in real applications not all the necessary knowledge is known a-priori, but rather unexpected and previously unseen asynchronous event can be triggered by other elements of the environment (including the user), 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 can thus be used as outlined before in applications for user monitoring and training, but also in other event processing applications: for example, 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). We are working presently on a first aid triage assignment example; in particular, we are programming a MAS so as to assign a triage color to each arriving patient according to present conditions, clinical history, risk assessment and other factors.

References

1. Aielli, F., Ancona, D., Caianiello, P., Costantini, S., De Gasperis, G., Di Marco, A., Ferrando, A., Mascardi, V.: Friendly & kind with your health: Human-friendly knowledge-intensive dynamic systems for the e-health domain. In: Int. Conf. on Practical Appl. of Agents and Multi-Agent Systems. Comm. in Computer and Inf. Sc., Springer (2016) 15–26
2. Alferes, J.J., Brogi, A., Leite, J.A., Pereira L.M. : Evolving Logic Programs. Logics in Artificial Int, European Conference, JELIA 2002, Proceedings.
3. Baral, C.: Know. repres., reas. and decl. problem solv. Cambridge Univ. Press (2003)
4. Bordini, R.H., Braubach, L., Dastani, M., ElSeghrouchni, A.F., Gomez-Sanz, J., Leite, J., O'Hare, G., Pokahr, A., Ricci, A.: A survey of programming languages and platforms for multi-agent systems. Informatica (Slovenia) **30**(1) (2006)
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. Costantini, S.: Self-checking logical agents. In: 8th Latin American Works. LA-NMR 2012. Volume 911 of CEUR Workshop Proceedings., CEUR-WS.org (2012) 3–30 Invited Paper, Extended Abstract in Proc. of AAMAS 2013.
7. Costantini, S.: ACE: a flexible environment for complex event processing in logical agents. In: Engineering Multi-Agent Syst., Third International Works., EMAS 2015, Revised Selected Papers. Volume 9318 of LNCS., Springer (2015)
8. Costantini, S.: The DALI agent-oriented logic programming language: Summary and references 2016 (2016) `http://www.di.univaq.it/stefcost/info.htm`.
9. Costantini, S., De Gasperis, G.: Memory, experience and adaptation in logical agents. In: Management Intelligent Systems: Second Intl. Symposium, Proc. Advances in Intelligent and Soft Computing, Springer (2013)
10. Costantini, S., De Gasperis, G.: Runtime self-checking via temporal (meta-)axioms for assurance of logical agent systems. In: Proc. of LAMAS 2014, 7th Works. on Logical Aspects of Multi-Agent Systems, held at AAMAS 2014. (2014) 241–255
11. Costantini, S., De Gasperis, G., Nazzicone, G.: DALI multi agent systems framework (July 2016) DALI: `http://github.com/AAAI-DISIM-UnivAQ/DALI`.
12. Costantini, S., De Gasperis, G., Nazzicone, G.: DALI for Cognitive Robotics: Principles and Prototype Implementation. In: Yuliya Lierler and Walid Taha (Eds.), Practical Aspects of Declarative Languages - 19th International Symposium, PADL 2017, Proceedings. Lecture Notes in Computer Science 10137, Springer (2017) 152–162
13. Costantini, S., Dell'Acqua, P., Moniz Pereira, L.: Conditional learning of rules and plans by knowledge exchange in log. agents. In: Proc. of RuleML 2011. LNCS 6826, Springer (2011)
14. Costantini, S., Dell'Acqua, P., Pereira, L.M., Toni, F.: Towards a model of evolving agents for ambient intelligence. In: Proc. of the Symposium on "Artificial Societies for Ambient Intelligence (ASAmI'07). (2007)
15. Costantini, S., DellAcqua, P., Tocchio, A.: Expressing preferences declaratively in logic-based agent languages. In: Proc. of Commonsense07, the 8th International Symposium on Logical Formalizations of Commonsense Reasoning, AAAI Press (2007) Event in honor of the 80th birthday of John McCarthy.
16. Costantini, S., Formisano, A.: Modeling preferences and conditional preferences on resource consumption and production in ASP. J. of of Alg. in Cognition, Inf. and Logic **64**(1) (2009).
17. Costantini, S., Riveret, R. : Event-action modules for complex reactivity in logical agents. Proceedings of AAMAS 2013, 13th Intl. Conf. on Autonomous Agents and Multi-Agent Systems.
18. Costantini, S., Tocchio, A.: A logic programming language for multi-agent systems. In: Logics in Artif. Int., Proc. of the 8th Europ. Conf. JELIA 2002. LNAI 2424, Springer (2002)
19. Costantini, S., Tocchio, A.: The DALI logic programming agent-oriented language. In: Logics in Artif. Int., Proc. of the 9th Europ. Conf. JELIA 2004. LNAI 3229, Springer (2004)
20. Costantini, S., Tocchio, A., Verticchio, A.: Communication and trust in tha Dali logic programming agent-oriented language. Intelligenza Artificiale, J. of Italian Association **2**(1) (2015) in English.
21. D'Inverno, M., Fisher, M., Lomuscio, A., Luck, M., de Rijke, M., Ryan, M., Wooldridge, M.: Formalisms for multi-agent systems. Knowledge Eng. Review **12**(3) (1997) 315–321
22. Fisher, M., Bordini, R.H., Hirsch, B., Torroni, P.: Computational logics and agents: a road map of current technologies and future trends. Comput. Int. J. **23**(1) (2007) 61–91
23. Fitzpatrick, P., contributors: YARP - Yet Another Robot Platform (May 2015) YARP: `http://github.com/robotology/yarp`
24. Leite, J.A.: Evolving knowledge bases: specification and semantics. Frontiers in Artificial Intelligence and Applications, vol. 81, 2003.
25. Metta, G., Fitzpatrick, P., Natale, L. : YARP: yet another robot platform. International Journal on Advanced Robotics Systems **3**(1) (2006) 43–48
26. Metta, G., Natale, L., Nori, F., Sandini, G., Vernon, D., Fadiga, L., Von Hofsten, C., Rosander, K., Lopes, M., Santos-Victor, J., et al.: The iCub humanoid robot: An open-systems platform for research in cognitive development. Neural Networks **23**(8) (2010) 1125–1134
27. Paauwe, P.A., Hoorn, J.F., Konijn, E.A., Keyson, D.V.: Designing Robot Embodiments for Social Interaction: Affordances Topple Realism and Aesthetics. Int. J. on Social Robotics **7**(5) (2015) 697–708
28. Paauwe, R.A., Keyson, D.V., Hoorn, J.F., Konijn, E.A.: Minimal requirements of realism in social robots: Designing for patients with acquired brain injury. In: Proc. of the 33rd Annual ACM Conf. on Human Factors in Computing Systems, ACM (2015) 2139–2144
29. Van Kemenade, M., Konijn, E.A., Hoorn, J.F.: Robots humanize care - moral concerns versus witnessed benefits for the elderly. In Verdier, C., Bienkiewicz, M., Fred, A.L.N., Gamboa, H., Elias, D., eds.: Proc. of HEALTHINF 2015, SciTePress (2015) 648–653