# Integrating Robot Control into the AgentSpeak(L) Programming Language

**Rodrigo Wesz**[1]

[1]Programa de Pós-Graduação em Ciência da Computação
Pontifícia Universidade Católica do Rio Grande do Sul (PUC-RS)
Porto Alegre – RS – Brazil

rodrigo.wesz@acad.pucrs.br

***Abstract.*** *Multiple software frameworks have been created to help developers model robot applications. These frameworks use low-level programming constructs suitable to control hardware components, such as sensors and actuators, but are limited in abstracting complexity. Conversely, agent programming languages support the implementation of agents using a higher level of abstraction, but these languages have been mostly restricted to the development of software agents. In this paper, we outline an architecture and programming constructs that integrate an agent programming language with a robot development framework in order to program autonomous robots using a higher-level abstraction. The resulting programming environment aims to facilitate the modeling of complex behaviors on robots using the abstraction of autonomous cognitive agents.*

## 1. Introduction

Programming robots is a complex activity since a number of challenges must be overcome: the limitations of an embedded system (e.g. limited battery, processing power and memory); the concurrency for the use of shared resources such as sensors and actuators and, at a high level, the challenge of making the robot perceive its current state and then choose actions to be performed in order to reach a goal. To help developers program mobile robots several *robot development frameworks* are available such as Carmen [Montemerlo et al. 2003], MOOS [Newman 2006] and ROS [Quigley et al. 2009]. These frameworks are able to handle sensors and actuators, but are limited when the use of *autonomous behaviour* is needed. One of the most common models for developing autonomous agents is based on the theory of human practical reasoning developed by Michael Bratman [Bratman 1987]. This computational model is called Beliefs, Desires and Intentions (BDI) Architecture and provides a computational analogy for the human practical reasoning characteristics.Several *agent programming languages (APLs)* exist that allow BDI agents to be programmed, such as AgentSpeak(L) [Rao 1996], Jason [Bordini et al. 2007] and 3APL [Hindrijs et al. 1999]. Through the use of these languages, we are able to simulate beliefs, desires and intentions at the software level, creating agents that have some characteristics, such as perception of the environment, knowledge acquisition and autonomous decision.

We aim to use agent programming languages architecture and their programming constructs to develop mobile robots using a higher-level abstraction. There are two advantages for choosing APLs to program mobile robots: first, the development process of any

software using a higher-level abstraction is simpler and has better maintainability compared to the use of a low-level programming language; second, these languages follow the BDI model, a model that tries to reproduce the human reasoning process as proposed by Bratman [Bratman 1987]. Other researchers have also studied the integration of robot programming constructs into agent programming languages, such as Ziafati [Ziafati 2013] [Ziafati et al. 2013], Verbeek [Verbeek 2003] and Wei [Wei and Hindriks 2013] and there is still work to be done in this area. This paper is a proposal for the integration of ROS and Jason using Cartago and the resulting programming environment aims to facilitate the modeling of complex behaviors on robots using the abstraction of autonomous cognitive agents.

This paper is organized as follows: In Section 2, we present a brief description of ROS, Jason, Cartago and JaCa Android. Section 3 presents the reason why we choose ROS and Jason, and a general description of the integration between them. In Section 4, we present some related work and finally, we conclude the paper in Section 5.

## 2. Background

### 2.1. ROS

The Robot Operating System (ROS) [Quigley et al. 2009] is a framework that aims to help software developers create applications for robots. It is composed of nodes, messages, topics, and services: *nodes* are software modules, namely processes that perform computation; *messages* are the data changed by nodes; *topics* are responsible to handle the communication between nodes: a node that is interested in a certain kind of data subscribes to the appropriate topic and a node expose its own data outside itself by publishing it to a given topic; and *services* are an alternative communication method in ROS, used for synchronous transactions.

### 2.2. Jason, Cartago and JaCa Android

Jason [Bordini et al. 2007] is an implementation of AgentSpeak(L) in Java. AgentSpeak(L) [Rao 1996] is a logical programming language for the implementation of BDI agents based on a restricted first-order language with events and actions. It can be viewed as a simplified textual language of Procedural Reasoning System (PRS) [Rao et al. 1992], one of the first implemented systems to be based on a BDI architecture and created to support real-time malfunction-handling and diagnostic systems [Weiss 1999]. PRS is implemented in LISP and provides goal-oriented behaviour and reactive behaviour.

CArtAgO (Common ARTifact infrastructure for AGents Open environments) [Ricci et al. 2009] is a framework used for engineering multi-agent applications, providing designer with the artifact notion and all the related concepts, such as object-oriented and service-oriented abstractions, autonomous activities (typically goal / task oriented) and structures (typically passive and reactive entities which are constructed, shared and used by the agents) [Ricci et al. 2009]. In CArtAgO, the sensors and actuators are structures that agents can create and use to partition and to control the information flow perceived from artifacts.

Artifacts are runtime objects that provide some kind of function or service which agents can use in order to achieve their objectives. Artifacts have a function description that can be read by the agents in order to discover the characteristics of a given artifact

and a set of operating instructions. Therefore, agents are able to interact with the artifacts by invoking operations and then observing the events generated from them. Namely, CArtAgO provide a natural way to model object-oriented and service-oriented abstractions (objects, components, services) at the agent level of abstraction [Ricci et al. 2009] through the use of artifacts.

JaCa is an agent-oriented programming platform comprising the integration of Jason and CArtAgO. JaCa uses Jason as programming language to develop and execute the agents and CArtAgO as the framework to develop and execute the environments [Santi et al. 2010]. In order to make JaCa language support mobile computing, a porting platform was developed on top of Android, where a mobile application can be realised as a workspace in which Jason agents are used to encapsulate the logic and the control of mobile application tasks, and artifacts are used as tools for agents to exploit available Android components and services [Santi et al. 2010]. Namely, JaCa Android is the porting of JaCa on Android, extended with a predefined set of artifacts specifically designed for exploiting Android functionalities [Santi et al. 2010]. According to Santi [Santi et al. 2010], the motivation to use JaCa on Android is address issues such as concurrency, asynchronous interactions with different kinds of services, and mostly, handle behaviours governed by specific context information (a typical characteristic of mobile applications).

## 3. A Proposal for the Integration of Jason and ROS

We decided for the use of ROS in this paper because of some important characteristics, such as: it is free and open source, thus we are able to change from the core code of ROS to implementations made by other researchers; it is very used by the academic community to program mobile robots, allowing us to have a large variety of tools already implemented and a good technical support; it supports cross-language development, allowing us to use Java (needed by Jason) with C (the ROS native language) and it has support for a great number of sensors and actuators. We use Jason to take advantage of some characteristics, such as: easily extensibility using Java; support for all the most important elements of the BDI architecture; support of strong negation, making available both closed-world and open-world assumptions and support for inclusion of additional information in beliefs and plans, allowing the creation of elaborated selection functions. Jason supports the creation of new internal actions, allowing developers to extend current Jason functionalities. We aim to use this feature in order to extend the agent internal capabilities and allow the agent to use legacy code.

In this paper, we propose the integration between Jason and ROS by creating a model that connects Jason primitives and ROS topics, extending the design of CArtAgO artifacts to support ROS features and behaviours, in order to program autonomous robots using a higher-level abstraction. This connection between the ROS low-level with the Jason high-level must be transparent to the mobile robot developer and the use of artifacts will help the achievement of this goal. The integration should support signatures and updates on the ROS topics and the conversion of the data from the ROS topics to a data understood by Jason (reciprocally, Jason's data must be converted to a data understood by ROS). Namely, we aim at expanding Jason agent programming language beyond the development of multi-agent systems, improving it to a high level mobile robot programming language. Inspired on JaCa Android proposal [Santi et al. 2010], which integrated

JaCa with Android, extending the agent language with a predefined set of artifacts specifically designed for exploiting Android functionalities, we aim to use Jason and CArtAgO to help the process of modeling the intermediate abstraction layer responsible to translate the Jason abstraction into ROS low-level robot control directives.

### 3.1. Proof of Concept

In order to verify the technical feasibility of implementing our proposed integration, an initial model was designed, with a basic set of primitives in Jason and two ROS topics. This proof of concept is a simple API, divided in two layers: the Java layer and the Jason layer, as shown in Figure 1. *Java layer* is basically composed by two modules: perceive and act. The perceive module is responsible for subscribing on the correct ROS sensor topics and to collect information from the sensors in order to forward this information as beliefs to Jason. The act module is responsible for publishing information from Jason into the ROS actuator topics. *Jason layer* is composed of a set of sensor primitives and a set of actuator primitives. The sensor primitives are connected to odom (nav_msgs/Odometry) topic and base_scan (sensor_msgs/LaserScan) topic, and the actuators primitives are connected just with odom ROS topic. Finally, a simple environment was created using Stage [Vaughan et al. 2003], composed by a robot free to "walk" on a hall and able to avoid obstacles (walls).

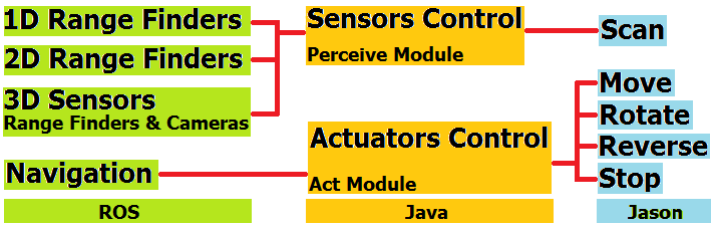


**Figure 1. The ROS layer, the Java layer and the Jason layer (our initial API)**

### 3.2. A proposal for the Integration Architecture

For a better understanding of how the integration of Jason and ROS works through the use of CArtAgO, we first briefly introduce how a common artifact is created. Listing 1 shows the Java pseudo-code of a CArtAgO artifact. We start importing the CArtAgO libraries (Line 1) and our artifact needs to extend Artifact class (Line 2). Inside the method init (Line 3), we need to define a property to be observable with the use of the artifact. The code continues with the definition of the artifacts operations (Line 6 and above). Details about how the operations works is beyond the scope of this paper.

For the purpose of modeling the intermediate abstraction layer between Jason and ROS, we propose to change the way in which the artifacts are created in order to make it supports ROS communication mechanisms. We propose the development of a new class called RosArtifact, that extends Artifact class from CArtAgO, adding features specifically designed for exploiting ROS robot sensors and control directives. Through the use of this class, the artifacts will be able to publish and to subscribe on ROS topics, thus the Jason agents are able to interact with ROS through the use of them. JaCa Android proposal [Santi et al. 2010] works in the same way: extending the Artifact class from CArtAgO

```
1  import cartago.*;
2  public class ClassName extends Artifact {
3   void init(){
4     defineObsProperty(...)
5     ...
6    }
7    @OPERATION void operationName(){
8    ...
9    }
10   ...
11 }
```

**Listing 1:** CArtAgO artifact pseudo-code sample.

and using it to define the artifacts that support control and observation of Android view interface.

We are developing a number of ROS artifacts, including: Movement, BaseIRScan and LaserScan, and we use artifact Movement as an example throughout this paper[1] Listing 2 shows the Java pseudo-code of a ROS artifact. We import ROS libraries (Line 2) and we create the ROS artifact extending RosArtifact class (Line 4). We use the method `subscribeRos()` from the mother class to subscribe the artifact on the ROS topic. A pseudo-code of `subscribeRos()` is shown on Listing 3. The sample artifact controls the movement of a robot, so we need to create the variables that will save linear and angular data to subscribe on the ROS topic: `linearData` (Line 8) and `angularData` (Line 9). Linear (Line 14), angular and odometer operations need to be created to control the data flow and to publish on ROS topic.

```
1  import cartago.*;
2  import ros.*;
3
4  RosClass.OdomData odom = new RosClass.OdomData(); //this will hold odometry data from ROS
5
6  public class Movement extends RosArtifact {
7    void init(string agentName, string rosTopic, string rosType){
8      subscribeRos(agentName, string rosTopic, string rosType); //this will subscribe on ROS topic
9      defineObsProperty("odom", odom);
10   }
11 @OPERATION
12 void linear(string direction, string velocity) { //handle linear odometry info from ROS topic
13   if ( direction != null && velocity != null ) {
14     RosClass.geometry_msgs.Twist twist = RosClass.publisherCmdVel.newMessage();
15     if (direction == "X"){ twist.getLinear().setX(velocity); }
16     ... // do the same for Y and Z
17     }
18   }
19 ... //TODO: angular and odometer operations
20 }
```

**Listing 2:** ROS artifact pseudo-code sample.

Listing 3 is the proposal of a pseudo-code of how the RosArtifact class subscribe on a ROS topic (Line 3). On this example, we are connecting on a IR sensor. We add a listener on the ROS topic thus any information from the sensor can be handled by our class (Line 4). All the details about how this class connects on a ROS topic is beyond the scope of this paper. In our proposal, the RosArtifact class should controls all the data flow between Jason and ROS. This is needed to avoid the classic bounded-buffer problem if ROS produce much more data than Jason could consume.

[1] Space limitations prevented us from detailing the other artifacts.

```
1  public void SubscribeRos(String agentName, String rosTopic, String rosType) {
2    ...
3    subscriberScan = m_rosnode.getConnectedNode().newSubscriber("/"+agentName+rosTopic, rosType+"._TYPE"
         );
4    subscriberScan.addMessageListener(new MessageListener<sensor_msgs.LaserScan>() {
5      @Override
6      public void onNewMessage(sensor_msgs.LaserScan message) {
7        m_scan = message; //holds odometry data
8      }
9    });
10   ...
11 }
```

**Listing 3:** ROS subscription on a sensor pseudo-code sample.

## 4. Related Work

Other researchers have investigated the integration of robot development frameworks and APLs. Ziafati et al [Ziafati et al. 2013] presents four requirements for BDI-based agent programming languages to facilitate the implementation of autonomous robot control systems; Ziafati [Ziafati 2013] identified and addressed BDI-based agent programming languages requirements for autonomous robot programming. The author developed an environment interface for 2APL to facilitate its integration with ROS; Verbeek [Verbeek 2003] research extends the 3APL language for high level robot control by creating a communication interface between the 3APL and ROS. He conducted experiments in a simulated and in a physical environment; and Wei [Wei and Hindriks 2013] presents a cognitive robot control architecture using GOAL [Hindriks 2009] and URBI [Baillie 2005].

## 5. Conclusion

In this paper, we propose the integration of an agent programming language and a robot development framework namely, the Jason implementation of AgentSpeak(L) and ROS. This integration of ROS and Jason using CArtAgO is a work in progress and is currently being implemented. We investigated the technical feasibility of implementing our proposed intermediate layer between Jason and ROS through a proof of concept implementation (presented in Section 3.1). In our proof of concept, communication between Jason and ROS is achieved using a ROS node extending Jason AgArch class and using ROS Java libraries. Our current work is on the codification of the complete design, which composes not only Jason and ROS, but the use of CArtAgO to reach our proposal objectives: to facilitate the modeling of complex behaviors on robots using the abstraction of autonomous cognitive agents.

**Acknowledgements** .
The author would like to acknowledge the guidance provided in this paper to Felipe Meneguzzi who supervises his MSc, and provided invaluable advice in the writing of this paper. I would like to thank the anonymous WESAAC reviewers for their valuable and constructive suggestions.

## References

Baillie, J.-C. (2005). Urbi: Towards a universal robotic low-level programming language. In *On 2005 IEEE RSJ International Conference on Intelligent Robots and Systems*, pages 820–825. IEEE.

Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007). *Programming multi-agent systems in AgentSpeak using Jason*, volume 8. Wiley. com.

Bratman, M. E. (1987). Intention, plans, and practical reason.

Hindrijs, K. V., de Boer, F. S., van der Hoek, W., and Meyer, J.-J. C. (1999). Agent programming in 3apl. In *Autonomous Agents and Multi-Agent Systems*.

Hindriks, K. V. (2009). Programming rational agents in goal. In *Multi-Agent Programming: Languages, Tools and Applications*, pages 119–157. Springer.

Montemerlo, D., Roy, N., and Thrun, S. (2003). Perspectives on standardization in mobile robot programming: the carnegie mellon navigation (carmen) toolkit. *Intelligent Robots and Systems, 2003. (IROS 2003)*, 3:2436 – 2441.

Newman, P. M. (2006). Moos - mission orientated operating suite. Department of Engineering Science Oxford University.

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., and Ng, A. (2009). Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3.

Rao, A. S. (1996). Agentspeak(l): Bdi agents speak out in a logical computable language. In *Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world*, MAAMAW '96, Secaucus, NJ, USA. Springer-Verlag New York.

Rao, A. S., Georgeff, M., and Ingrand, F. (1992). An architecture for real-time reasoning and system control. In *Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world: agents breaking away*, pages 34–44.

Ricci, A., Piunti, M., Viroli, M., and Omicini, A. (2009). Environment programming in cartago. In *Multi-Agent Programming: Languages, Tools and Applications*, pages 259–288. Springer US.

Santi, A., Guidi, M., and Ricci, A. (2010). Jaca-android: An agent-based platform for building smart mobile applications. *Third International Workshop, LADS 2010*.

Vaughan, R. T., Gerkey, B. P., and Howard, A. (2003). On device abstractions for portable, reusable robot code. *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2121 – 2427.

Verbeek, M. (2003). 3apl as programming language for cognitive robots. Master's thesis, Utrecht University.

Wei, C. and Hindriks, K. V. (2013). An agent-based cognitive robot architecture. In *Programming Multi-Agent Systems*, pages 54–71. Springer.

Weiss, G. (1999). *Multiagent Systems: A Modern Approach to Distributed Modern Approach to Artificial Intelligence*. Massachusetts Institute of Technology.

Ziafati, P. (AAMAS 2013). Programming autonomous robots using agent programming languages. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems*.

Ziafati, P., Dastani, M., Meyer, J.-J., and van der Torre, L. (2013). Agent programming languages requirements for programming autonomous robots. In *Programming Multi-Agent Systems*, pages 35–53. Springer.