Using a BDI Agent to Represent a Human on the Factory Floor of the ARIAC 2023 Industrial Automation Competition*

Leandro Buss Becker[1,5], Anthony Downs[2], Craig Schlenoff[2], Justin Albrecht[2], Zeid Kootbally[2], Angelo Ferrando[3], Rafael Cardoso[4], and Michael Fisher[1]

[1] University of Manchester, Manchester, UK
{leandro.bussbecker,michael.fisher}@manchester.ac.uk
[2] National Institute of Standards and Technology, MD, USA
{anthony.downs,craig.schlenoff,justin.albrecht,zeid.kootbally}@nist.gov
[3] University of Genova, Genova, Italy
angelo.ferrando@unige.it
[4] University of Aberdeen, Aberdeen, UK
rafael.cardoso@abdn.ac.uk
[5] Federal University of Santa Catarina, Florianópolis, Brazil

Abstract. The "Agile Robotics for Industrial Automation Competition" (ARIAC) is an international robotic competition carried out in a simulated factory floor using ROS 2 (Robot Operating System)/Gazebo. Competitors control one gantry robot, four AGVs, and many other elements/devices, overcoming a range of agility challenges in this simulated environment, and are provided with a scoring system to evaluate their performance during the tasks. This paper describes one of the agility challenges in ARIAC 2023, which pertains to a simulated human operator on the factory floor. In undertaking manufacturing tasks, competitors must avoid close proximity between the gantry robot and the human not to get penalized. The human operator is implemented as a Belief-Desire-Intention (BDI) agent in Jason. It is provided with a range of different potential types of behaviour in what concerns with how such human reacts when in proximity to the gantry robot. Three different personalities are presented, ranging from a minimally intrusive up to a very intrusive one. A preliminary analysis was conducted to evaluate the impact of using the developed Jason agent in the ARIAC 2023 competition.

Keywords: Robots in human environments · BDI agents · Jason/ROS

1 Introduction

Organised by the National Institute of Standards and Technology (NIST) since 2017, the Agile Robotics for Industrial Automation Competition[6] [5] (ARIAC)

* This work was supported by NIST in the USA and the UK's Royal Academy of Engineering through its Chair in Emerging Technologies scheme.

[6] https://www.nist.gov/ariac

is an annual simulation-based competition which brings together researchers and practitioners to tackle challenges related to agile robotics that industry is facing.

ARIAC 2023 uses version 2 of the Robot Operating System (ROS 2), an open-source framework that offers a comprehensive set of libraries and tools to develop robot software, in conjunction with Gazebo, a physics-based simulator. Together, ROS 2/Gazebo provide a flexible and efficient platform for designing, testing, and deploying robotics applications.

One of the main goals of ARIAC is to provide real-life manufacturing scenarios where humans and robots share a low-volume high-mix workload in a collaborative environment. As such, a new challenge has been introduced in ARIAC 2023, which consists of avoiding close contact between a human operator that moves around the factory floor making inspections and the robots present in the workcell. The workcell contains the following robots: (i) four AGVs that move forward or backward within a given straight lane, and (ii) one gantry[7] robot that consists of a manipulator mounted onto an overhead system that allows it to move along the entire factory floor.

It is our aim that the human operator can have different types of behaviours (from now on we refer to these types as personalities), varying the level of interference caused by the human operator to the gantry robot. The human operator must also attempt not to collide against the AGVs while they move within the factory floor. Figure 1 depicts the ARIAC 2023 simulation scenario.

It is required for the gantry robot not to get closer to the human operator than established in the ISO/TS 15066:2016 standard "Robots and robotic devices – Collaborative robots", which addresses the safety issue of robot speed and separation monitoring [6]. A similar restriction also applies for the AGVs. Competitors get penalized if such restrictions are not properly followed.

This paper concerns the implementation of the movement control strategy for the human operator. Given that Belief-Desire-Intention (BDI) agents [9] can emulate the cognitive reasoning of humans in a very natural way, this paradigm was selected to be used for controlling our human operator. More specifically, the implementation of our BDI agent is done in Jason [1], a well-known BDI programming language [2]. The challenge that we face is this work relates not simply with implementing the Jason agent, but also with how to properly integrate it within the complex ARIAC 2023 simulation environment. Moreover, it is also a challenge how to guide competitors so that they can properly deploy all the tools needed to run our agent within the simulation environment.

The main contributions of this paper can be summarized are as follows: (i) we describe how to integrate the Jason BDI agent for controlling the human operator within the complex ARIAC 2023 simulation scenario; (ii) we detail how such agent is in fact programmed in Jason and how it interfaces with ROS 2/Gazebo. (iii) we analyze the impact of using such Jason agent in ARIAC 2023 from the final user (competitors) perspective, in what concerns the deployment and usage difficulties and also the impact in respect to the CPU utilization.

[7] Gantry robots are also called Cartesian or Linear robots. In the ARIAC 2023 documentation it will be also referred to as ceiling robot.

The remaining parts of this paper are organised as follows. Section 2 describes the ARIAC competition. Section 3 presents the software architecture of our solution. The developed Jason agent is detailed in Section 4. Our conclusions are presented in Section 5.

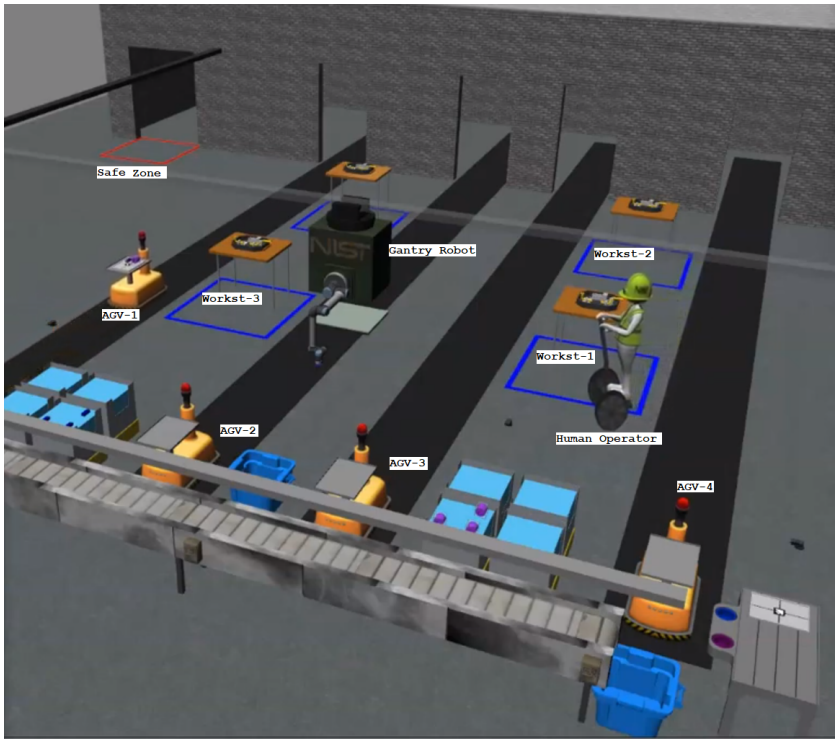

Fig. 1. The ARIAC 2023 scenario. The red square in the top left (safe zone) is the starting position of the human operator. The four blue squares below the tables are the workstations. AGV-1 is moving and the other three are stationary. The human operator is facing the gantry robot, but from the image it is not possible to guess its current personality: if *antagonistic* or *indifferent* it will move towards the gantry; if *helper* it will turn around and move back to the workstation-2.

2 Agile Robotics for Industrial Automation Competition

ARIAC is an annual competition which aims to tackle challenges that industry is facing in agile robotics. The main goal of ARIAC is to test the agility of industrial robot systems and to enable industrial robots on shop floors to be more productive, more autonomous, and to require less time from shop floor workers. In ARIAC, agility is defined broadly to address: (1) task failure identification and

recovery by robots, (2) automated planning to minimise (or eliminate) the up-front robot programming time when a new task is introduced, and (3) operation in fixtureless environments, where robots can sense the environment and perform tasks on parts that are not in predefined locations. The competition participants are required to develop a robot control system for a gantry robot in order to perform kitting operations in a simulated environment.

Prior to designing ARIAC in 2017, NIST explored existing robotics competitions to ensure none of them already addressed industrial robotics agility. The Amazon Picking Challenge [4] was one of the competitions related to challenges addressed in ARIAC. The competition assessed the capability of robots to perform some of the common pick and place operations that are currently performed by humans. The Robot Perception Challenge [7] was another competition which was relevant to agility challenges. The goal of this competition was to drive improvements in sensing and perception technologies for next-generation robots. ARIAC was designed to test and measure Industrial Robot Agility in a holistic sense, because no other competitions were covering that niche.

Figure 1 depicts the simulated environment where the ARIAC 2023 competition takes place. The gantry can move in the simulated environment to interact with objects in order to perform kitting for assembly tasks (announced dynamically during the simulation). A kit is an order for specific items, which can be found on shelves, on the conveyor belt, and in bins. The robot builds kits by picking up all the required items and placing them into one of the trays located on the automated guided vehicles (AGVs). When an order is completed, the AGV delivers the kit and a final score is given to the participants' systems. The final score takes into account many aspects, such as if the type/colour of the selected item matches the type/colour required by the order; the accuracy of products' pose in the tray; and the time taken by the control system to complete a kit (measured in simulation seconds).

The ARIAC 2023 competition has eight "agility challenges"[8]. They represent extra difficulties that competitors may face while performing kitting tasks. For example, competitors could face faulty and/or flipped parts to assemble. Challenges are sampled together in different "trials" that the competitors must overcome during the qualification and final rounds of the competition. Within the scope of this paper we focus on the "human operator" agility challenge.

2.1 Human Operator Agility Challenge

This challenge consists of inserting a human operator that navigates through the factory floor (workcell). In Figure 1, it is possible to observe the presence of the human operator (on the right) facing the gantry robot (on the left). The goal of this challenge is to test the ability of the competitors' control system for the gantry robot to avoid collisions with the human operator, otherwise it will incur a penalization.

[8] `https://ariac.readthedocs.io/en/latest/competition/challenges.html`

The simulated human operator will take one of the three personalities in a given trial. Note that, once a personality has been selected for a trial, it will not change during that trial. Even though the development and integration of changing a personality at runtime in ARIAC would be feasible, to simplify the evaluation of the competitor's controller, we opted for a static agent's personality. Regardless of the personality that the agent adopts, it was decided to avoid random moves and to make simplistic, predefined, movement patterns along the four workstations that simulate working/inspection tasks, something common for humans to do within a factory floor. The human operator agent will keep travelling to these workstations and working until the trial ends.

If the human operator and one of the robots get closer than a minimum safety distance (details for the calculation are provided in the next section), then the human is teleported to a safe zone (the top left position shown in Figure 1). Exceptionally, the human operator is not teleported if it gets close to a non-moving (static) AGV. Moreover, if the teleport operation is caused for being too close to the gantry robot, then the competitor team gets penalised, which also implies disabling the gantry robot for 8 seconds; afterwards, the normal operation is resumed. In such case the human operator is teleported away purely to give time to the competitors to recover and to avoid situations where the human can behave too aggressively and keep the gantry in a deadlock.

The agent's personalities are as following:

1. *Indifferent*: The human operator follows a predetermined path, regardless of the location of the robots in the environment.
2. *Antagonistic*: The human operator purposefully moves towards the gantry robot to interfere with the robot's current task.
3. *Helpful*: The human operator will move to another workstation (changing direction to avoid the gantry) once the gantry robot is detected to be at a certain distance (`safety distance` $\times 2$).

The *helpful* agent was designed to be minimally intrusive, and should rarely interfere the competition. On the other hand, the *antagonistic* agent is intended to be very intrusive, and is likely to frequently cause penalization to the competitors. We foresee that the *indifferent* agent is the one that will better judge the competitors' skills to avoid contact with the human operator.

2.2 Safety Distance Calculation

The safety distance between the human operator and the robots (gantry robot and AGVs) is derived from the ISO/TS 15066:2016 standard - "Robots and robotic devices - Collaborative robots", which addresses the safety issue of robot speed and separation monitoring [6]. ISO/TS 15066:2016 specifies that the minimum allowable distance between a robot and a human is

$$d_{min} = k_H(t_1 + t_2) + k_R t_1 + B + \delta$$

where t_1 is the maximum time between the actuation of the sensing function and the output signal switching devices to the off state, t_2 is the maximum

response time of the machine (i.e., the time required to stop the machine), δ is an additional distance, based on the expected intrusion toward the critical zone prior to actuation of the protective equipment, k_H is the speed of the intruding human, k_R is the speed of the robot, and B is the Euclidean distance required to bring the robot to a safe, controlled stop.

3 Simulation Software Overview

Figure 2 illustrates the elements within the simulation scenario that are of interest for the developed BDI agent: the human operator, the four AGVs, and the gantry robot. The relevant related information about such elements – mainly location and speed – must be constantly updated within the agent, which can only actuate towards the human operator. The additional elements in the scene (shown in Figure 1) are treated simply as obstacles that should be avoided by the navigation control algorithm running in ROS 2.

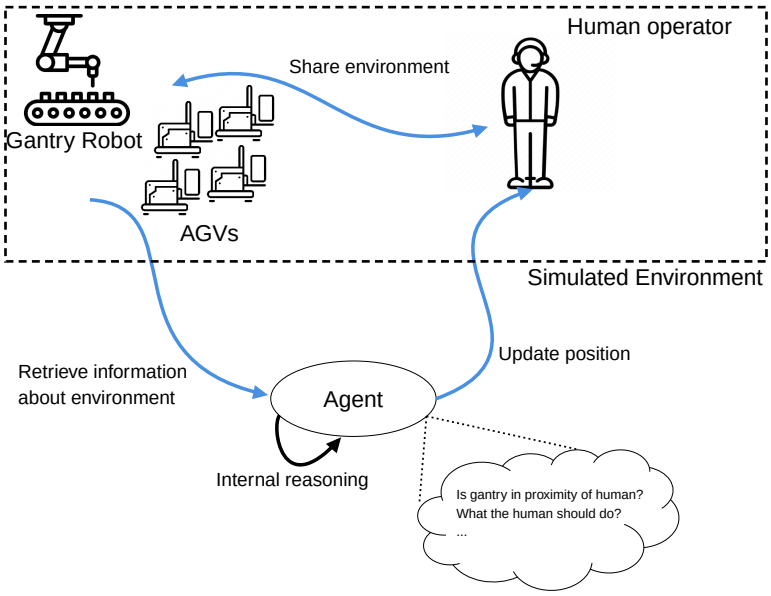

Fig. 2. Overview of the simulation environment from the BDI agent perspective.

A relatively complex software architecture was built to support this simulation environment. Such software architecture is composed of several elements that include, mostly, artifacts from ROS 2 (nodes, topics, services, actions, plugins) and the Jason agent. Figure 3 depicts the elements of the proposed solution[9].

[9] Source code available at `https://github.com/usnistgov/ARIAC`

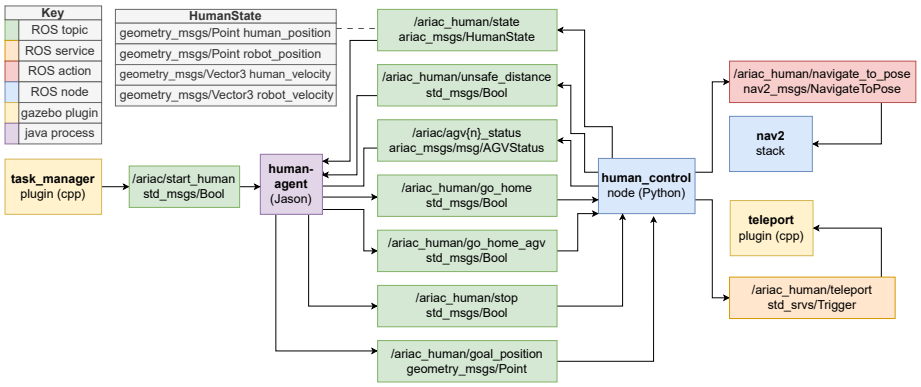

Fig. 3. Software architecture artifacts that support the adopted simulation scenario.

Analysing Figure 3 from left to right, first there is the *task_manager* Gazebo plugin. It is in charge of initialising all the components that constitute the competition scenario. It is also in charge of publishing the */ariac/start_human* topic to start our Jason's *human–agent*, which was already launched but remains idle until a message on this topic is received. Continuing to the right of the figure, the agent can publish to the three topics at the bottom and subscribe to the two topics at the top, which are all related to the *human_control* ROS 2 node. This node also interacts with the *teleport* Gazebo plugin and with the *navigation stack* (part of the ROS 2 distribution).

4 The Human Agent

The Jason agent is in charge of the high-level control of the human operator. In the simulation, the human is represented as a robot with a human mesh on top of it. Representing the human as a robot allows the human to easily interact with other ROS elements in the simulation. The agent is responsible for controlling the movements of the human, calling ROS 2 functions such as $move(x, y)$ and $stop()$. It must also be constantly updated about the position of the gantry robot and the AGVs, so that it can properly reason about the actions to be taken. It was decided that the human behaviour should be simple and predictable to a certain extent, i.e., there should be no random moves. Therefore, in general terms, the human operator must move around four predefined points of interest within the virtual factory's shop floor (the workstations). The default movement occurs in a clockwise basis starting at workstation 4 ($4 > 2 > 1 > 3 > 4 > \ldots$).

Jason programs are implemented separately into agent and environment programs. Agent programs consist of (in this order): initial beliefs and rules; initial goals; and plans. Plans are written with traditional AgentSpeak syntax [8] `triggering_event : context <- body.` wherein the `triggering_event` can be the addition/deletion of a `belief` or a `goal`, the `context` are the preconditions of the

plan, and the `body` is a sequence of operations (`actions` or addition/deletion of `beliefs`/`goals`). Environment programs are written in Java and define the semantics of the actions that agents can execute, as well as providing the agent with environment perceptions.

4.1 Initial Beliefs and Initial Goal

The initial lines of code from the agent define a set of static beliefs that are used for orientation purposes. For instance, it defines the (x, y) coordinates of the four target positions (workstations 1 to 4), the first position for the robot to visit, and the order in which such positions should be visited (for either counterclockwise and clockwise movement directions).

Two beliefs can change at runtime: `working(Loc)` and `counterClockWise`. The first keeps track of the current station, so that the agent can derive the next target position; while the latter, if present in the agent's belief, indicates a counterclockwise movement direction (otherwise the agent adopts clockwise movement). The agent has one initial goal to wait for the *human_start* message.

4.2 Plans for Movement Control

The agent's main task is to keep the human operator moving through the predefined points. We implement this with two plans, with triggering events `+!work` and `+work_completed`, as shown in Listing 1.1. The first has a context used only to find the coordinates of the desired destination (ln.1)[10]. It then removes the belief that indicates the previous target location (ln.2), and sets the belief with the current target location (ln.3). Finally, it calls an external action in charge of activating the movement at the ROS node (ln.4). The `+work_completed` is triggered when the ROS node indicates that the human operator reached the target position. Its context is used to find the next location to be visited. There is an analogous version of this plan for the counterclockwise movement.

```
1 +!work(Loc) : location(Loc, X, Y, Z) <-
2     -working(_);
3     +working(Loc);
4     move(X, Y, Z).

6 +work_completed(_) : working(Loc) & next_loc(Loc,Next) &
    counterClockWise  <- !work(Next).
```

Listing 1.1. Main plans to move the human operator.

In the plan on Listing 1.2, the `+gantry_disabled(_)` belief is added when the gantry is disabled due the fact that the distance between the gantry and the human operator is violating the safety distance. This belief is added with a parameter for debugging purposes. Note the use of _ as in Prolog, which indicates that the term can be unified with anything (i.e., we do not care about its contents

[10] 'ln.' will be used as abbreviation for *line* throughout the paper.

in this plan). A similar plan was created for when the human operator is too close to an AGV. The difference in the AGV case is that it calls a teleport service that does not penalise the competitor.

```
1 +gantry_disabled(_) : firstStation(ST) <-
2     .drop_all_desires;
3     teleport_safe; // stop + teleport to safe zone
4     .wait(8000);
5     !!work(ST).
```

Listing 1.2. Plan for when the Gantry is disabled.

In such a plan, the agent drops its own desires (ln.2) using an internal action (Jason predefined actions that do not interact with the environment). This is done to stop all goals currently executed by the agent (e.g., moving to a workstation). Then, we call an external action (implemented in the environment) to teleport the human operator to the safe location (ln.3). This is obtained on the Gazebo side by means of a custom plugin (developed as part of the human challenge integration in ARIAC). After that, the agent waits a fixed amount of time (ln.4); the latter is domain specific and has been decided to give time to the gantry's controller to restore its own tasks. At the end, the plan concludes by calling the `!work` once more, and restoring the standard movement of the human operator in the simulation by going to the first workstation.

4.3 Implementing Personalities

The human personality defines how it behaves in respect to the gantry position. This role is defined upon the agent's initialisation based on the parameter specified in a particular trial (we expect that in ARIAC 2023 there will be at least one trial with each personality). As mentioned in section 2, the three possible personalities are *Indifferent*, *Antagonistic*, and *Helpful*.

In order to implement these three different personalities within the Jason agent, we provide three distinct implementations for the `+gantry_detected` perception. Each implementation lies in a different agent program file (*asl* extension), which is loaded according to the agent initialisation parameter. This perception is triggered when the human operator and the gantry get "too close". This distance, which is computed in Jason's *Environment* class, is defined as being twice the safety distance (calculated as shown in Section 2.2).

The implementation for the *indifferent* personality is proforma, as in fact it has no condition and does not take any action (it is just an empty plan). Therefore we only discuss here the implementations for the *antagonistic* and the *helpful* personalities, as follows.

The core part of *antagonistic* agent behavior is shown in Listing 1.3.

```
1 +gantry_detected(_) :
2         working(Loc) & next_loc(Loc,Next) <-
3     stop_movement;
4     .drop_all_desires;
5     move_to_gantry;
6     .wait("+work_completed(_)");
7     !!work(Next).
```

Listing 1.3. Jason code for the agent with the *antagonistic* personality.

It has a context that will always be `true` since it uses beliefs that are always supposed to be present in the belief base, but it is needed in order to allow identifying the destination that the human is currently moving to (ln.2). It first stops and cancels any navigation goal (ln.3), then it drops all desires (ln.4) and triggers an external action requiring the human to move towards the gantry (ln.5). Afterwards the plan remains blocked until it reaches the target position (ln.6). When this holds, it resumes moving to the the next station (ln.7).

The core part of the *helpful* agent implementation is shown in Listing 1.4. It requires two distinct plans because it can be moving in either clockwise or counterclockwise directions. The agent keeps the internal belief `counterClockWise` , which is used in the plan contexts to reason about the current direction. If this belief is present (condition in ln.2), then the movement is counterclockwise, and the plan in ln.1–6 is triggered. Otherwise, if it is absent (condition in ln.9), the movement is clockwise, triggering the plan in ln.8–13. Besides having different contexts, each of them adjusts the direction in a different way (ln.5 versus ln.12) and resumes the movement towards a different destination (ln.6 versus ln.13).

```
1  @detected[atomic]
2  +gantry_detected(_) : working(Loc) & previous_loc(Loc,Prev)
       & counterClockWise <-
3      stop_movement;
4      .drop_all_desires;
5      -counterClockWise;
6      !!work(Prev).
7
8  @detectedCounter[atomic]
9  +gantry_detected(_) : working(Loc) & next_loc(Loc,Next) &
       not counterClockWise <-
10     stop_movement;
11     .drop_all_desires;
12     +counterClockWise;
13     !!work(Next).
```

Listing 1.4. Jason code for the agent with the *helpful* personality.

The plans for the *helpful* agent are implemented as `atomic`, a predefined plan annotation available in Jason (@id[atomic] where id is a unique plan identifier) to stop considering concurrent intention stacks (i.e., only the intentions related to this plan can be selected). This is required because we do not want these

plans to be interrupted while executing, otherwise the agent could lose track of its current movement direction.

4.4 The Environment Class

Jason's `Environment` class is responsible for performing the agents' interaction with the external world. In this case, the `Environment` class is responsible for subscribing to the ROS topics of interest and transforming the messages within them into perceptions for the agent. It is also responsible for implementing the agent's external actions, which in this case means publishing on ROS topics. The previous Figure 3 presented the topics-of-interest for our human agent.

Our implementation is based in the ROS-A interface[11] [3] which makes use of the ROSBridge[12] library. Listing 1.5 shows our `RosEnv` class definition and its `init()` method, where subscriptions to ROS topics are defined (e.g., ln.8–26) and, when received, are transformed into perceptions for the agent (ln.21–23). In total, the agent subscribes to four ROS topics, as depicted in the Figure 3.

The method `executeAction()` is responsible for decoding the required external action, as presented in ln.1–11 of Listing 1.6. An example of ROS–topic publication is given in ln.12–15. In total, the agent publishes four different ROS topics, as also depicted in the Figure 3.

4.5 Results and Additional Remarks

This section presents the preliminary analysis conducted to evaluate the impact of using the developed Jason agent in the ARIAC 2023 competition – a complete analysis should be done once the competition is finished. Such analysis is performed in terms of deployment and usage difficulties – from the final users (competitors) perspective – and also in respect to the impact on the computing resources utilization.

The metric used to evaluate the users difficulties regards the number of related issues opened in the competition's Github [13]. From a total of 256 issues opened until the present moment, only 3 (1.2%) were related with the "human operator" agility challenge: #221, #229, and #245. The first issue was related with installation problems of two required artifacts, Java and ROS 2: (i) wrong JDK version, and (ii) missing ROS 2 *nav2-simple-commander* package. The issue #229 addressed the effects the human in proximity with the AGVs, and triggered some internal parameters tuning in our software. The last issue addressed difficulties for running the system within a Docker package.

[11] `https://github.com/rafaelcaue/jason-rosbridge`
[12] http://wiki.ros.org/rosbridge_suite
[13] https://github.com/usnistgov/ARIAC/issues?q=is%3A+issue

```
public class RosEnv extends Environment{
 RosBridge bridge = new RosBridge();
 ...
 @Override
 public void init(String[] args) {
    super.init(args);
    bridge.connect("ws://localhost:9090", true);
    bridge.subscribe(SubscriptionRequestMsg.generate("
    /ariac_human/state")
     .setType("ariac_msgs/msg/HumanState")
     .setThrottleRate(1)
     .setQueueLength(1),
    new RosListenDelegate() {
     public void receive(JsonNode dt, String st) {
      MessageUnpacker<HumanState> unpkr = new
          MesageUnpacker<HumanState>(HumanState.class);
      HumanState m = unpkr.unpackRosMessage(dt);
      gpX = m.robot_position.x; //store Gantry position
      ... //same to y,z
      double dis_robotHuman = calcDistanceRH(m);
      double safe_dis     = calcSafeDistance(m);
      if(dis_robotHuman>2*safe_dis){
       Literal gDet=new LiteralImpl("gantry_detected");
       gDet.addTerm(new NumberTermImpl(ctrDt++));
       addPercept("human",gDet);
    }  }  }
    ); // END subscribe "/ariac_human/state"
 ... //continue subscription to other ROS topics
 } // END init()
```

Listing 1.5. *RosEnv* Jason's Environment with ROS–topics subscription.

Performance tests were conducted to evaluate the impact of the developed BDI-agents in respect to the computing resources utilization. Such tests were executed using a Linux Ubuntu 20.04 workstation with an Intel Core i9-10920X CPU with 24 cores at 3.50 GHz, 64 GiB of memory, and the NVIDIA GeForce RTX 3080 graphics card. ROS 2 Galactic was used. The *ps* command at 2 s intervals was used to log CPU utilization. The performance data was collected by running a 275 s long experiment. As the experiment script is launched it spawns 27 processes related with ROS 2/Gazebo and one process related with the Jason agent. For the ROS 2/Gazebo processes, the average CPU utilization was 510% (five cores entirely plus 10% of a sixth core). For the Jason process, the average CPU utilization was 6.5%. The Jason CPU usage decreased slightly over time. Overall, Jason required only 1.27% of the CPU portion used by ROS 2/Gazebo, which shows that it does not provide a significant overhead when observing the complete simulation system.

```
1  public boolean executeAction(String ag, Structure ac){
2    if (ac.getFunctor().equals("move")) { //
3         ... //get x,y,z "terms" from ac
4         move(x,y,z);
5    } else if (ac.getFunctor().equals("stop_movement"))
6         stop_moving();
7    ... //continue with other ext. actions
8    else return false;
9    informAgsEnvironmentChanged();
10   return true; // action successfully executed
11 }  ... //here starts the method's implementation
12 public void stop_moving() {
13   Publisher pub = new Publisher("/ariac_human/stop",
         "std_msgs/Bool", bridge);
14   pub.publish(new Bool(true));
15 }
```

Listing 1.6. External actions and ROS topic publishers.

We recorded videos demonstrating the three different human personalities in action in the competition environment.[14] As the gantry is stopped close to the station 1, only the indifferent human will in fact reach this station – and then will continue moving up to the point that it gets teleported. The antagonistic human will change its direction towards the gantry before reaching station 1, and shortly after it will also get teleported. The helpful human will turn around as it gets close to the station 1 and will continue moving in the opposite direction.

5 Conclusions and Future Work

This paper presented what is considered to be the first use of BDI agents in the ARIAC competition. Amongst the challenges that we faced when implementing this agent, we highlight the high-level of complexity involving the software architecture of the ARIAC 2023 competition. Our Jason agent was required to interact with different components of the simulation environment, so that it could properly control the human operator in the simulation. Besides the Jason agent, including its environment, it was necessary to implement a couple of additional ROS 2/Gazebo components, such as the Python ROS node for movement control and the CPP Gazebo plugin to support the teleport operation.

The conducted analysis presented evidences that using BDI technologies did cause significant overhead to the final users in terms of complexity for properly putting the system to run. More importantly, it did not lead to a significant overhead in terms of CPU utilization.

Even though the currently developed movement control for the human operator is simplistic if considering the full capacities of a BDI application, it serves as basis for more sophisticated/complex versions that will come in the future.

[14] Indifferent: `https://youtu.be/5pqm5WSQNTw`. Helpful: `https://youtu.be/7CH4skoOs8c`. Antagonistic: `https://youtu.be/TQh9GQ1BbFw`.

This can, therefore, be seen as a successful initiative, which can also be observed as a pedagogical action towards evangelising the use of cognitive/BDI agents within non-agents developer communities, such as the robotics one, which is the community mostly involved with the ARIAC competition. We also understand this to be an initial seed towards spreading the use of cognitive agents within industrial automation environments.

As future work, we aim to analyse the practical effects (consequences) on competitors in the human challenge after ARIAC 2023 takes place, in especial in what concerns the impact of the three different personalities of the human operator.

References

1. Bordini, R.H., Hübner, J.F., Wooldrige, M.: Programming Multi-Agent Systems in AgentSpeak using *Jason*. John Wiley & Sons (2007)
2. Cardoso, R.C., Ferrando, A.: A Review of Agent-Based Programming for Multi-Agent Systems. Computers **10**(2), 16 (2021)
3. Cardoso, R.C., Ferrando, A., Dennis, L.A., Fisher, M.: An Interface for Programming Verifiable Autonomous Agents in ROS. In: Multi-Agent Systems and Agreement Technologies. pp. 191–205. Springer (2020)
4. Correll, N., Bekris, K.E., Berenson, D., Brock, O., Causo, A., Hauser, K., Okada, K., Rodriguez, A., Romano, J.M., Wurman, P.R.: Analysis and Observations from the First Amazon Picking Challenge (2016)
5. Harrison, W., Downs, A., Schlenoff, C.: The Agile Robotics for Industrial Automation Competition. AI Magazine **39**(4), 73–76 (2018)
6. Marvel, J.A.: Performance Metrics of Speed and Separation Monitoring in Shared Workspaces. IEEE Trans. on Automation Science and Eng. **10**(2), 405–414 (2013)
7. Marvel, J.A., Hong, T.H., Messina, E.: 2011 solutions in perception challenge performance metrics and results. In: Proc. of the Workshop on Performance Metrics for Intelligent Systems. p. 59–63. ACM, New York, NY, USA (2012)
8. Rao, A.S.: AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. In: Agents Breaking Away, MAAMAW 1996. LNCS, vol. 1038, pp. 42–55. Springer (1996)
9. Rao, A.S., Georgeff, M.P.: BDI Agents: From Theory to Practice. In: Lesser, V.R., Gasser, L. (eds.) Proceedings of the First International Conference on Multiagent Systems. pp. 312–319. The MIT Press, United States (1995)