BusAI: An BDI-agent-based Urban Transport Information System

Lucas Lira, Andrei Serafim, Nilson Lazarin, Bruno Freitas, Carlos Pantoja

Bacharelado em Sistemas de Informação
Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (Cefet/RJ)
Nova Friburgo, RJ – Brazil

chon@grupo.cefet-rj.br

Abstract. *This paper proposes an information system for urban transport, leveraging an Embedded Multi-agent System based on the Belief-Desire-Intention (BDI) model. It involves developing a simulated environment where AI agents, implemented using Jason and communicating via Javino, manage real-time bus data and interact with virtual information points. This system aims to enhance the urban mobility experience by providing passengers with accurate, dynamic information regarding bus location, occupancy, and estimated arrival times, thereby improving operational efficiency and passenger satisfaction. This paper details a proof of concept demonstrating the viability of this multi-agent approach in addressing real-world public transport challenges.*

1. Introduction

The contemporary urban public transport scenario is characterized by growing challenges, intensified by accelerated urbanization and the demand for more efficient and transparent services (Vuchic 2017). A vast portion of the population depends on public transport, and the absence of accurate real-time information about vehicle location, occupancy, and waiting times at bus stops can culminate in operational inefficiencies and negatively impact user experience (Olio et al. 2017). In various urban contexts, passenger flow management and operational data communication persist as manual processes or are dependent on legacy systems, which frequently prove inadequate to keep pace with the complex and changing dynamics of the urban environment (Giaretta and Di Giulio 2017).

This work proposes the development of an Intelligent Passenger Information System for Urban Transport, designed to reduce the communication gap between users and transport services by providing real-time operational information. The goal is to enhance passenger experience and optimize service efficiency by making data such as bus location and estimated arrival available through digital interfaces at bus stops. The system is implemented using the Jason platform (Bordini et al. 2007), which supports the Belief-Desire-Intention (BDI) agent model (Rao and Georgeff 1995), and the Javino framework (Lazarin and Pantoja 2015) to interact with a physical environment.

The rest of this paper is organized as follows: Section 2 presents the necessary background to understand the proposal. Section 3 presents the system's agent architecture. Section 4 presents the proposal implementation. Section 5 presents a proof of concept of the system. Finally, Section 6 presents the conclusion.

2. Background

Multi-agent Systems (MAS) represent an area of Artificial Intelligence dedicated to the study of systems composed of multiple intelligent agents that interact with each other and with the environment to achieve individual or collective objectives (Bordini and Vieira 2003). An agent, in this context, can be defined as an autonomous, proactive, reactive, and social software entity. Autonomy refers to the agent's ability to operate without direct human intervention; proactivity manifests in its ability to take initiatives to achieve goals; reactivity concerns its capacity to respond to changes in the environment; and sociality denotes the ability to interact with other agents. The application of MAS is particularly advantageous in complex and dynamic scenarios, where control decentralization and collaboration between autonomous entities can lead to more robust and adaptable solutions (Balaji and Srinivasan 2010).

The Belief-Desire-Intention (BDI) model (Bratman et al. 1988) constitutes one of the most influential cognitive architectures for intelligent agents. Inspired by commonsense psychology, BDI structures an agent's reasoning around three primary mental components: *Beliefs*: represent the agent's knowledge about the current state of the environment, other agents, and itself. These are the pieces of information the agent considers true. *Desires*: correspond to the objectives or world states the agent aspires to achieve. These are the agent's motivations. *Intentions*: are the agent's commitments to execute specific plans to satisfy its desires. Intentions are the action plans the agent commits to follow (Bordini et al. 2007).

The BDI dynamics involve a reasoning cycle where the agent perceives the environment, updates its beliefs, revises its desires, and, based on its intentions, selects and executes action plans. This model offers a structured and intuitive way to conceive the behavior of intelligent agents in complex environments. Jason provides a complete environment for executing, debugging, and monitoring AgentSpeak(L) agents, making it a robust tool for MAS development (Bordini et al. 2007).

For agents to interact with the physical environment or with simulations that represent this environment, an efficient communication mechanism is essential. Javino (Lazarin and Pantoja 2015) acts as a communication bridge, allowing the exchange of messages and data between Java applications (where Jason agents are executed) and microcontrollers (such as Arduino) or simulation environments that emulate hardware behavior (Freitas et al. 2025). This bidirectional communication is fundamental for agents to receive perceptions from the ”world” (simulated sensor data, for example) and send actuation commands to modify this ”world” (e.g., display information on a simulated user interface).

3. System Agent Architecture

In the context of the Intelligent Passenger Information System, the agent architecture is distributed, with specialized agents performing distinct but collaborative roles. Essentially, the system will be composed of:

Bus Agent: This agent will be responsible for representing the Bus entity in the simulated environment. Its beliefs will include dynamic data such as current location (latitude and longitude), number of passengers, and maximum capacity. The Bus Agent will aim to keep these beliefs updated and proactively communicate its state to relevant agents, emulating the functionality of an embedded agent capable of perceiving its simulated physical environment.

Bus Stop Agent: This agent will represent a specific bus stop and its respective passenger information system. Its perceptions will involve user clicks requesting destinations and location and occupancy information from buses. The Bus Stop Agent will have objectives to receive and process passenger requests, consult information from available buses, calculate distances and estimated arrival times, and display these options on the user interface (virtual totem). It will act as an intelligent intermediary, coordinating information between buses and passengers.

Coordinator Agent: To ensure fluidity and integrity of communication between Bus Agents and Bus Stop Agents, we introduce a third fundamental element in our architecture: the Coordinator Agent. Imagine a scenario with dozens of buses and bus stops. If each one tried to talk to all others simultaneously, we would create a true information ”traffic jam”. A Bus Stop Agent could receive conflicting data, and a Bus Agent would be overloaded with multiple requests. This is where the Coordinator Agent comes in, acting as the orchestrator of these data exchanges.

4. Prototyping and Implementation

This section details the prototyping and implementation process of the Intelligent Passenger Information System, focusing on the development environment, agent creation, and simulation of interactions with the environment. Initially, the system was developed and tested in a controlled and simulated environment, running directly on a computer. This approach allows validating agent logic and software component integration before an eventual transition to a real application scenario.

The interaction between the agents from the architecture is fundamental to the system dynamics. For example, the Bus Stop Agent can ”ask” Bus Agents through the Coordinator Agent about their location and number of passengers, using inter-agent communication facilitated by the Jason environment. Integration with Javino allows the Java simulation environment to provide perceptions to agents (e.g., simulated bus movement, passenger boarding/alighting) and receive commands from agents (e.g., display data on the totem interface).

Development Environment Configuration: The sensor firmware was specifically designed for the Arduino platform and tested within the SimulIDE (SimulIDE Project 2025) simulation environment, acting as the vehicle’s sensor module. It is based on three main libraries: *Javino.h*, which manages structured communication with the agent system; *TinyGPS++.h*, for efficiently decoding GPS location data; and *SoftwareSerial.h*, which creates a dedicated communication port for the GPS module, freeing up the main hardware port. In the setup() function, the digital pins are configured as inputs to read the passenger flow sensors (entry on pin 2, exit on pin 3) and the mechanical failure alert button (pin 4). The *javino.perceive(getPercepts)* instruction is a key design point: it registers the *getPercepts* function as a callback, meaning the function that prepares data for sending will only be executed when the host system requests the information.

The main loop() is lean and non-blocking, continuously delegating the tasks of reading the sensors and the GPS. The core logic resides in the getPercepts() callback function. It builds a message with ”percepts” that include:

- Static identifiers (device(bus)): Discrete events, such as passenger(in) or passenger(out), which are sent only once per occurrence.
- The vehicle’s state, reporting busStatus(stopped) in case of a failure or busStatus(running) for normal operation.
- GPS data, which is crucially added to the message only if considered valid by the library, ensuring data integrity.

```
#include <SoftwareSerial.h>
#include <TinyGPS++.h> //https://github.com/mikalhart/TinyGPSPlus
#include <Javino.h> // https://javino.chon.group
Javino javino;
SoftwareSerial gpsSerial(10, 11); // RX 10, TX 11
TinyGPSPlus gps;
boolean in, out, stop = false;

void setup() {
 pinMode(2, INPUT); pinMode(3, INPUT); pinMode(4, INPUT);
 gpsSerial.begin(9600);
 javino.perceive(getPercepts); javino.start(9600);
}

void serialEvent(){javino.readSerial();}

void loop(){javino.run(); getRawData();gpsReader();}

void getRawData(){
 if(digitalRead(2) == HIGH) in = true;
 if(digitalRead(3) == HIGH) out = true;
 if(digitalRead(4) == HIGH) stop = true;
   else stop=false;
}

void getPercepts(){
 javino.addPercept("dest(1)"); javino.addPercept("device(bus)");
 if(in){javino.addPercept("passager(in)"); in=false;}
 if(out){javino.addPercept("passager(out)"); out=false;}
 if(stop) javino.addPercept("busStatus(stopped)");
 else{
  javino.addPercept("busStatus(running)");
  if(gps.location.isValid()) {
   javino.addPercept("lat("+String(gps.location.lat(), 6)+")");
   javino.addPercept("lng("+String(gps.location.lng(), 6)+")");
  }
 }
}

void gpsReader() {
  while (gpsSerial.available() > 0) {gps.encode(gpsSerial.read())
      ;}
}
```

Listing 1. Bus device firmware.

```
#include <Javino.h> // https://javino.chon.
      group
Javino javino;

const byte pins[] = {2,3,4,5,6,7,8,9,10,11};
const int btnN = sizeof(pins)/sizeof(pins[0]);
boolean destinations[btnN] = {false};

void setup() {
 for (int i=0; i<btnN; i++)
  pinMode(pins[i], INPUT);
 javino.perceive(getPercepts);
 javino.start(9600);
}

void serialEvent(){javino.readSerial();}
void loop(){javino.run(); btnPressed();}

void btnPressed() {
 for (int i = 0; i < btnN; i++)
  if(digitalRead(pins[i]) == HIGH)
   destinations[i] = true;
}

void getPercepts() {
 javino.addPercept("device(busPoint)");
 for (byte i = 0; i < btnN; i++)
  if(destinations[i])
   javino.addPercept("dest("+ String(i+1)+")");
 memset(destinations, 0, sizeof(destinations));
}
```

Listing 2. Bus point device firmware

Thus, the code represents a robust and reactive telemetry node that efficiently collects and formats data from the simulated environment to be processed by the ”Bus Agent”.

The Role of the ARGO Agent: Listing 3 details the reasoning of the bus agent. This agent (ARGO (Pantoja et al. 2016)) is responsible for monitoring the internal state of the vehicle. Its main logic is triggered by ”passenger entering” (passenger(in)) and ”passenger leaving” (passenger(out)) events, which are perceived through the serial port. Upon perceiving a passenger’s entry, the agent updates the system’s state by decreasing the number of available seats (seats(L-1)). Conversely, upon perceiving a departure, it increases the number of seats (seats(L+1)). This is a direct translation of a physical event (the movement of a person) into a digital state update within the system.

Listing 4 illustrates the reasoning of the bus stop agent. The main function of this agent is to act as the initial interface for the user at the bus stop. When a user selects a destination (+*dest(V)*), the agent does not process the information by itself. Instead, it initiates the communication chain within the artificial intelligence system. It sends a message (*.send*) to another agent (*agentPonto*) with the objective (achieve) of obtaining information (*get.info(V)*) about the requested destination. This demonstrates how the agent translates a user action into a digital message that triggers other components of the system to fulfill the request.

```
busStatus(A,B,C,D,E) :- busStatus(A) & lat(B) & lng(C) &
       seats(D) & dest(E).
seats(45).
serialPort(ttyEmulatedPort0). /* using simulIDE */
//serialPort(ttyUSB0). /* using Arduino Board */

!start.

+!start: serialPort(Port) <-
 .port(Port); .limit(500);  .percepts(open).

+passager(in) <- ?seats(L); -+seats(L-1).
+passager(out) <- ?seats(L); -+seats(L+1).
```

Listing 3. Reasoning of the bus ARGO agent in AgentSpeak(L).

```
serialPort(ttyEmulatedPort1). /* using simulIDE */
//serialPort(ttyUSB1). /* using Arduino */

!start.

+!start: serialPort(Port) <-
  .port(Port);
  .limit(1000);
  .percepts(open).

+dest(V) <-
  .print("Requesting info to ",V);
  .send(agentPonto, achieve, getInfo(V)).
```

Listing 4. Reasoning of the point ARGO agent in AgentSpeak(L).

```
/*beliefs*/
iotGateway("skynet.chon.group",5500).
myUUID("0a99ff05-1308-41a3-a752-173224698233").
/* intentions */
!connect.
/* plans */
+!connect: iotGateway(Server,Port) & myUUID(ID) <-  .connectCN(Server,Port,ID).

+!busStopped[source(UUID)] <- .abolish(fleet(UUID,_,_,_,_,_)).

+!busRunning(Status,Latitude,Longitude,Seats,Destination)[source(UUID)] <-
    .abolish(fleet(UUID,_,_,_,_,_)); +fleet(UUID,Status,Latitude,Longitude,Seats,Destination).
```

Listing 5. Reasoning of the Coordinator agent in AgentSpeak(L)

```
/* beliefs */
iotGateway("skynet.chon.group",5500).
myUUID("ba0cd5ba-70ab-49b7-90b3-2bedac71a8da").
controllerUUID("0a99ff05-1308-41a3-a752-173224698233").
/* intention */
!start.
/* plans */
+!start: iotGateway(Server,Port) & myUUID(ID) <-
 .connectCN(Server,Port,ID); !getStatus.

+!getStatus <-
  .abolish(busStatus(_,_,_,_,_));
  .broadcast(askOne,busStatus(A,B,C,D,E));
  .wait(busStatus(_,_,_,_,_),5000);
  !info; .wait(10000); !getStatus.
-!getStatus <- !info; .wait(10000); !getStatus.

+!info: busStatus(S,Lat,Lng,Seats,Dest) <-
 ?controllerUUID(C);
 .sendOut(C,achieve,busRunning(S,Lat,Lng,Seats,Dest)).
+!info: not busStatus(S,Lat,Lng,Seats,Dest) <-
 ?controllerUUID(C);
 .sendOut(C,achieve,busStopped).
-!info.
```

Listing 6. Reasoning of the bus Communicator agent in AgentSpeak(L).

```
/* beliefs */
iotGateway("skynet.chon.group",5500).
myUUID("0e95bdbe-39eb-4bff-a43c-10a83caa723d").
controllerUUID("0a99ff05-1308-41a3-a752-173224698233").
/* intentions */
!connect.
/* plans */
+!connect: iotGateway(Server,Port) & myUUID(ID) <-
  .connectCN(Server,Port,ID).
+!getInfo(D)<-
 -+destination(D); -+attempt(5); !askController.

+!askController: destination(D) & controllerUUID(C) &
      not fleet(_,_,_,_,_,D) & attempt(T) & T>0<-
 .sendOut(C,askAll,fleet(_,running,_,_,_,D));
 -+attempt(T-1); .wait(1000); !askController.
+!askController: fleet(_,_,_,_,_,D) & attempt(T) & T=5<-
 .abolish(fleet(_,_,_,_,_,_)); !askController.
+!askController: destination(D) & attempt(T) & T=0 <-
 .print("Without information about",D).
-!askController.

+fleet(A,B,C,D,E,F): destination(Dest) & F=Dest <-
 .print(E," seats"," GPS Location ",C," ",D).
```

Listing 7. Reasoning of the point Communicator agent in AgentSpeak(L).

Communicator Agent Development: The **Bus**, **Coordinator**, and **Bus Stop** agents (Communicator (Nunes et al. 2018)) will be coded in AgentSpeak(L) (Codes 5-7), using corresponding *.asl* files, according to the conceptual modeling presented in the Backgroud section. Each agent's initial beliefs will be defined to establish its initial state and knowledge about the simulated environment. For example, Bus Stop agents will have beliefs about their simulated geographic location *(stop_location(my_stop_id, Lat, Lon))*.

Agent behavior logic will be implemented through plans that will react to specific events, such as perceiving a bus moving or a passenger clicking on the simulated information system interface. Jason's inter-MAS communication capability will be employed to allow bus stop to query the state of bus, over the *Chon IoT Network* (Lazarin et al. 2023).

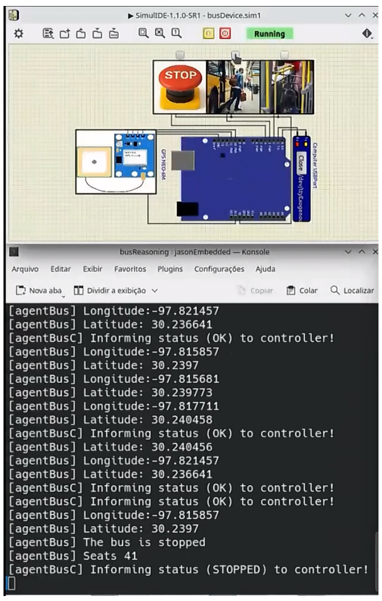
5. Proof of Concept

To validate the proposed agent architecture, we built a controlled simulation environment. This scenario was designed to mirror the fundamental interactions that would occur in a real transport system, focusing on agent communication and logic. In this scenario, we instantiated 1 agent of the following types:

Bus Agent: An agent to represent the bus, to send its data to the Coordinator Agent, in Figure 1a. In this experimental setup, the bus agent interacts with the system by means of a *stop*, *leave*, and *enter* buttons. Once a passenger enters or leaves the bus, it taps the appropriate button, updating the bus capacity. When the bus driver starts or stops the bus, he presses the "stop" button. It's important to note that, while the process here is manual, it could be improved by adding further mechanisms to automatically detect these events, which is not in the scope of this work.

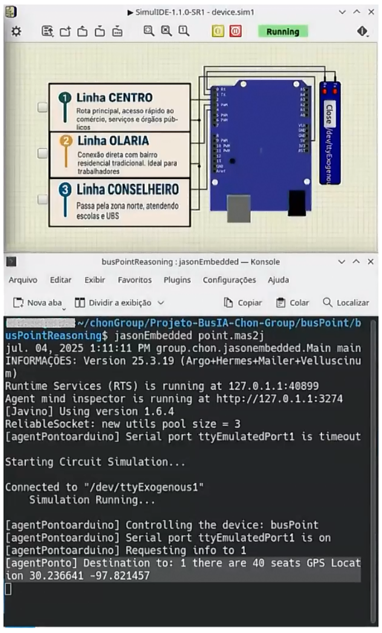
Bus Stop Agent: A simulated virtual bus stop where a user (through interface clicks) could request route information, in Figure 1b. Here, we have 3 bus lines: *CENTRO*, *OLARIA*, and *CONSELHEIRO*. Whenever a passenger wants to receive updates on the bus's route and capacity, he taps the appropriate button for the line requested.

Coordinator Agent: Receives information from the bus and forwards it to the bus stop as a communication "bridge" for better data handling in the Agent's mind, in Figure 1c.



```
[agentBus] Longitude:-97.821457
[agentBus] Latitude: 30.236641
[agentBusC] Informing status (OK) to controller!
[agentBus] Longitude:-97.815857
[agentBus] Latitude: 30.2397
[agentBus] Longitude:-97.815681
[agentBus] Latitude: 30.239773
[agentBus] Longitude:-97.817711
[agentBus] Latitude: 30.240458
[agentBusC] Informing status (OK) to controller!
[agentBus] Latitude: 30.240456
[agentBus] Longitude:-97.821457
[agentBus] Latitude: 30.236641
[agentBusC] Informing status (OK) to controller!
[agentBusC] Informing status (OK) to controller!
[agentBus] Longitude:-97.815857
[agentBus] Latitude: 30.2397
[agentBus] The bus is stopped
[agentBus] Seats 41
[agentBusC] Informing status (STOPPED) to controller!
```

(a) Bus agent


SimulIDE-1.1.0-SR1 - device.sim1
Running
Linha CENTRO
Rota principal, acesso rápido ao comércio, serviços e órgãos públicos
Linha OLARIA
Conexão direta com bairro residencial tradicional. Ideal para trabalhadores
Linha CONSELHEIRO
Passa pela zona norte, atendendo escolas e UBS
Close
/dev/ttyExogenous
busPointReasoning : jasonEmbedded — Konsole
Arquivo
Editar
Exibir
Favoritos
Plugins
Configurações
Ajuda
Nova aba
Dividir a exibição
Copiar
Colar
Localizar

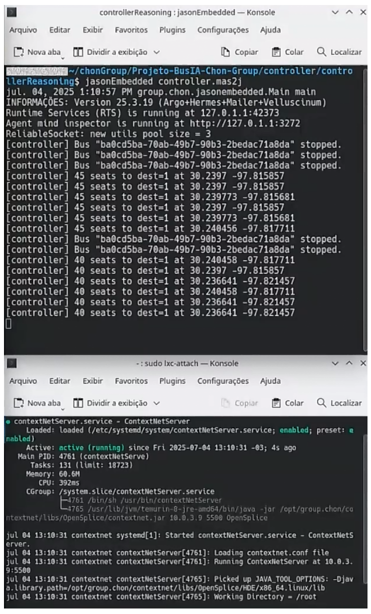
```
[redacted]~/chonGroup/Projeto-BusIA-Chon-Group/busPoint/b
usPointReasoning$ jasonEmbedded point.mas2j
jul. 04, 2025 1:11:11 PM group.chon.jasonembedded.Main main
INFORMAÇÕES: Version 25.3.19 (Argo+Hermes+Mailer+Velluscinu
m)
Runtime Services (RTS) is running at 127.0.1.1:40899
Agent mind inspector is running at http://127.0.1.1:3274
[Javino] Using version 1.6.4
ReliableSocket: new utils pool size = 3
[agentPontoarduino] Serial port ttyEmulatedPort1 is timeout

Starting Circuit Simulation...

Connected to "/dev/ttyExogenous1"
    Simulation Running...

[agentPontoarduino] Controlling the device: busPoint
[agentPontoarduino] Serial port ttyEmulatedPort1 is on
[agentPontoarduino] Requesting info to 1
[agentPonto] Destination to: 1 there are 40 seats GPS Locat
ion 30.236641 -97.821457
```

(b) Bus stop agent



```
[redacted]~/chonGroup/Projeto-BusIA-Chon-Group/controller/contro
llerReasoning$ jasonEmbedded controller.mas2j
jul. 04, 2025 1:10:57 PM group.chon.jasonembedded.Main main
INFORMAÇÕES: Version 25.3.19 (Argo+Hermes+Mailer+Velluscinum)
Runtime Services (RTS) is running at 127.0.1.1:42373
Agent mind inspector is running at http://127.0.1.1:3272
ReliableSocket: new utils pool size = 3
[controller] Bus "ba0cd5ba-70ab-49b7-90b3-2bedac71a8da" stopped.
[controller] Bus "ba0cd5ba-70ab-49b7-90b3-2bedac71a8da" stopped.
[controller] Bus "ba0cd5ba-70ab-49b7-90b3-2bedac71a8da" stopped.
[controller] 45 seats to dest=1 at 30.2397 -97.815857
[controller] 45 seats to dest=1 at 30.2397 -97.815857
[controller] 45 seats to dest=1 at 30.239773 -97.815681
[controller] 45 seats to dest=1 at 30.2397 -97.815857
[controller] 45 seats to dest=1 at 30.239773 -97.815681
[controller] 45 seats to dest=1 at 30.240456 -97.817711
[controller] Bus "ba0cd5ba-70ab-49b7-90b3-2bedac71a8da" stopped.
[controller] Bus "ba0cd5ba-70ab-49b7-90b3-2bedac71a8da" stopped.
[controller] 40 seats to dest=1 at 30.240458 -97.817711
[controller] 40 seats to dest=1 at 30.2397 -97.815857
[controller] 40 seats to dest=1 at 30.236641 -97.821457
[controller] 40 seats to dest=1 at 30.240458 -97.817711
[controller] 40 seats to dest=1 at 30.236641 -97.821457
[controller] 40 seats to dest=1 at 30.236641 -97.821457
```

```
● contextNetServer.service - ContextNetServer
     Loaded: loaded (/etc/systemd/system/contextNetServer.service; enabled; preset: e
nabled)
     Active: active (running) since Fri 2025-07-04 13:10:31 -03; 4s ago
   Main PID: 4761 (contextNetServe)
      Tasks: 131 (limit: 18723)
     Memory: 60.6M
        CPU: 392ms
     CGroup: /system.slice/contextNetServer.service
             ├─4761 /bin/sh /usr/bin/contextNetServer
             └─4765 /usr/lib/jvm/temurin-8-jre-amd64/bin/java -jar /opt/group.chon/co
ntextnet/libs/OpenSplice/contextnet.jar 10.0.3.9 5500 OpenSplice

jul 04 13:10:31 contextnet systemd[1]: Started contextNetServer.service - ContextNetS
erver.
jul 04 13:10:31 contextnet contextNetServer[4761]: Loading contextnet.conf file
jul 04 13:10:31 contextnet contextNetServer[4761]: Running ContexNetServer at 10.0.3.
9:5500
jul 04 13:10:31 contextnet contextNetServer[4765]: Picked up JAVA_TOOL_OPTIONS: -Djav
a.library.path=/opt/group.chon/contextnet/libs/OpenSplice/HDE/x86_64.linux/lib
jul 04 13:10:31 contextnet contextNetServer[4765]: Working Directory = /root
```

(c) Coordinator agent

Figure 1. Proof of concept.

5.1. Reproducibility

Aiming to provide the reproducibility of this paper, the source code of agents (using JasonEmbedded (Pantoja et al. 2023)) and simulator (using simulated exogenous approach (Freitas et al. 2025)), also a video demonstration is available[1].

6. Conclusion

This work presents the design and implementation of a multi-agent Passenger Information System for urban transport, utilizing the BDI (Belief-Desire-Intention) architecture. The system prototype was composed of three core agent types Bus, Bus Stop, and Coordinator responsible for sharing contextual data such as location and passenger count. The simulation demonstrated the feasibility of using decentralized agents to manage and disseminate real-time transport information.

In our proof-of-concept test, the system achieved an average message exchange success rate of 98% and an average response time of approximately 250 milliseconds per interaction between agents, even in a simulated environment. These results, while preliminary, indicate that the communication logic is stable and capable of handling frequent updates.

One of the main limitations of the current version is the use of simulated GPS coordinates. Due to the complexity of integrating real-time geolocation data and the scope of this first phase, we prioritized validating the communication logic and agent interactions. Another simplification was the manual ”stop” signal, which served as a placeholder for a future diagnostic integration.

In terms of scalability, the current architecture has not yet been stress-tested with dozens of buses and bus stops operating simultaneously. It is expected that, in such a scenario, communication traffic could increase significantly, requiring message optimization and possibly a hierarchical coordination strategy.

Compared to existing real-world Passenger Information Systems, our approach emphasizes decentralized decision-making and the possibility of richer contextual data exchange features that could make the system more resilient and adaptable in dynamic environments.

As future work, we aim to integrate the system with real-time GPS data using APIs such as Google Maps (Google 2025). We also propose the development of a Diagnostic Agent capable of interfacing with vehicle onboard systems to automatically detect failures and proactively inform users and operators. Additional enhancements include estimating arrival times based on traffic data, tracking bus capacity, and leveraging historical data to improve predictions and planning.

During the process, we identified the need to manage not only location but also the vehicle's operational status. The implementation of the manual ”stop” button, which allows the driver to signal a failure, was a first step in this direction. That way, passengers can be warned in real-time about unforeseen events, such as an engine failure, by consulting the panel at the bus stop.

[1]`https://papers.chon.group/WESAAC/2025/BusAI/`

References

Balaji, P. G. and Srinivasan, D. (2010). *An Introduction to Multi-Agent Systems*, pages 1–27. Springer, Berlin, Heidelberg. DOI: 10.1007/978-3-642-14435-6_1.

Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007). *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons.

Bordini, R. H. and Vieira, R. (2003). Linguagens de programação orientadas a agentes: uma introdução baseada em AgentSpeak(L). *Revista de informática teórica e aplicada*, 10(1):7–38. URL: `https://lume.ufrgs.br/handle/10183/19885`.

Bratman, M. E., Israel, D. J., and Pollack, M. E. (1988). Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4(3):349–355. DOI: 10.1111/j.1467-8640.1988.tb00284.x.

Freitas, B., Lazarin, N., Pantoja, C., and Viterbo, J. (2025). Integrating Simulated Exogenous Environments to Support the Learning Process of the Embedded MAS Approach. In *WEI 2025*, pages 1195–1206, Porto Alegre. SBC. DOI: 10.5753/wei.2025.9115.

Giaretta, J. B. Z. and Di Giulio, G. M. (2017). The role of the Information and Communication Technologies (ICT) in the urban 21st cend in the emergence of new social movements: reflections on experiences in the São Paulo megacity. *Revista Brasileira de Estudos Urbanos e Regionais*, 20(1):161. DOI: 10.22296/2317-1529.2018v20n1p161.

Google (2025). Maps platform documentation. `https://developers.google.com/maps`. Accessed: 2025-08-11.

Lazarin, N. and Pantoja, C. (2015). A Robotic-agent Platform For Embedding Software Agents using Raspberry Pi and Arduino Boards. In *Anais do IX Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações*, pages 13–20, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/wesaac.2015.33308.

Lazarin, N., Pantoja, C., and Viterbo, J. (2023). Towards a Toolkit for Teaching AI Supported by Robotic-agents: Proposal and First Impressions. In *Proceedings of XXXI WEI*, pages 20–29, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/wei.2023.229753.

Nunes, P., Almeida, I., Picanço, T., Pantoja, C., Samyn, L., Jesus, V., and Manoel, F. (2018). Exploring Communication Between Embedded Multi-Agent Systems in Smart Environments for IoT: A Proposed Laboratory. In *WESAAC 2018*, pages 238–243, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/wesaac.2018.33272.

Pantoja, C. E., Jesus, V. S. d., Lazarin, N. M., and Viterbo, J. (2023). A Spin-off Version of Jason for IoT and Embedded Multi-Agent Systems. In *Intelligent Systems*, pages 382–396, Cham. Springer. DOI: 10.1007/978-3-031-45368-7_25.

Pantoja, C. E., Stabile, M. F., Lazarin, N. M., and Sichman, J. S. (2016). ARGO: An Extended Jason Architecture that Facilitates Embedded Robotic Agents Programming. In *Engineering Multi-Agent Systems*, pages 136–155, Cham. Springer International Publishing. DOI: 10.1007/978-3-319-50983-9_8.

Rao, A. S. and Georgeff, M. P. (1995). BDI Agents: From Theory to Practice. In *Proceedings of ICMAS'95*, pages 312–319, San Francisco, CA, USA.

SimulIDE Project (2025). Simulide: Real time electronic circuit simulator. `https://simulide.com`. Accessed: 2025-08-11.

Vuchic, V. R. (2017). *Urban Transit: Systems and Technology*. John Wiley & Sons.

Olio, L., Ibeas, A., de Ona, J., and de Ona, R. (2017). *Public Transportation Quality of Service: Factors, Models, and Applications*. Elsevier.