Tampere University

AUT.842 Distributed and intelligent automation systems

Assignment 2

Valtteri Nikkanen	282688
Lauri Nuutinen	283219
Väinö Kurvi	274490

Table of contents

Introduction

Our assignment was to implement a conveyor system using multiagent systems, learn JADE, multi-agent interactions and FIPA (Foundation of Intelligent Physical Agents) messages and agent behaviours. In figure 1 is our test layout for the conveyor system.

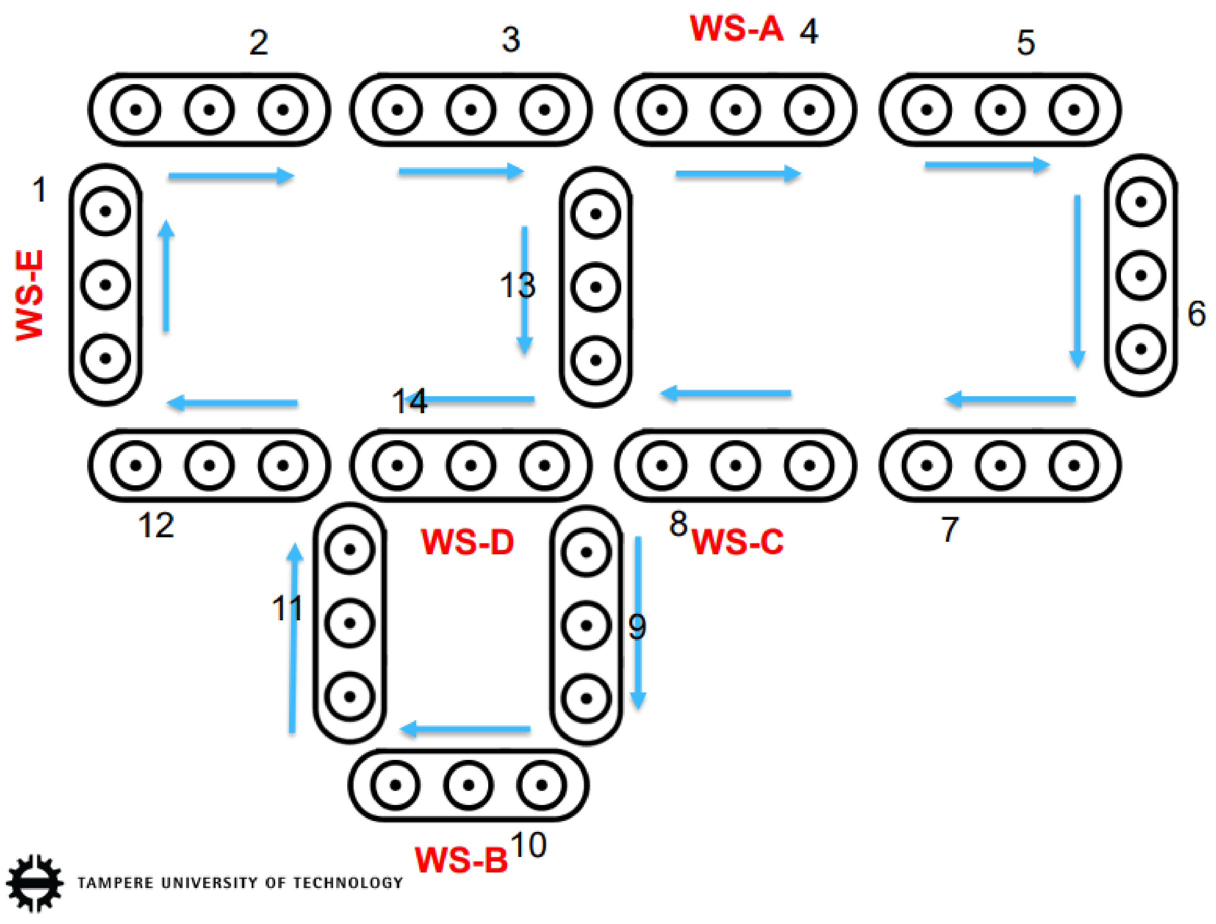


Figure 1: Test layout

Constraints and functional requirements for the conveyor system are:

- Pallets can be placed at any conveyor.
- Pallet's destination can be any conveyor.
- Layout can change.
- Conveyors can have three states:
    - Idle
    - Busy
    - Down
- Conveyors states can be modified through external messages.

Tasks

Task 1 was to implement a transfer control for the conveyor system. This includes the following:

- Moving a pallet from one conveyor to another
- Configurable conveyor transfer time
- Ability to provide pre-defined routes for the pallet.
- System being able to handle multiple pallets simultaneously.

Task 2 was to make a route finder that plans the route for the pallet. Route finder includes:

- Find path algorithm.
- minimizing number of conveyors

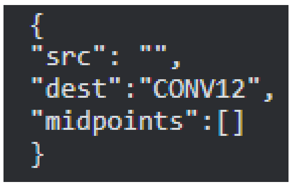
Bonus task mixes both task 1 and task 2 features together.

- Conveyor agents should be able to transfer pallets and reroute them if abnormal situation occur during the transfer
- Abnormal situations:
  - Conveyor down
  - Conveyor busy

Basic principle of our implementation

In our implementation the LayoutBuilderAgent creates hard coded layout of conveyor agents and gives them parameters which are name, class and what are its next neighbours. ConveyorAgent handles sending and receiving messages from other conveyors and route calculations.

Pallet can be placed on a conveyor by giving the conveyor a destination conveyor or the route for the pallet. When only the destination is given the conveyor executes a find route algorithm which calculates the shortest route by the number of conveyors it passes through to get to the destination. When the shortest route has been found the destination conveyor informs the source that the pallet can start to move. When a pallet is placed on a conveyor the conveyor sends a JSON message to its neighbouring conveyors like in image 2 that contains the name of the first conveyor, destination conveyor of the pallet and route how to get to the destination.



```
{
"src": "",
"dest":"CONV12",
"midpoints":[]
}
```

Figure 2: JSON message

When a pallet goes on a conveyor its state is changed to BUSY. ConveyorAgents states and pallet transfer times can also be changed by external messages.

If a conveyor that was on a route is down when the pallet tries to get there the conveyor, then runs the find route algorithm again. Instead, if the conveyor is busy the pallet waits on the previous conveyor until it can move. Conveyor system is also capable of handling multiple pallets at once. Use case diagram of our implementation is presented in figure 3.

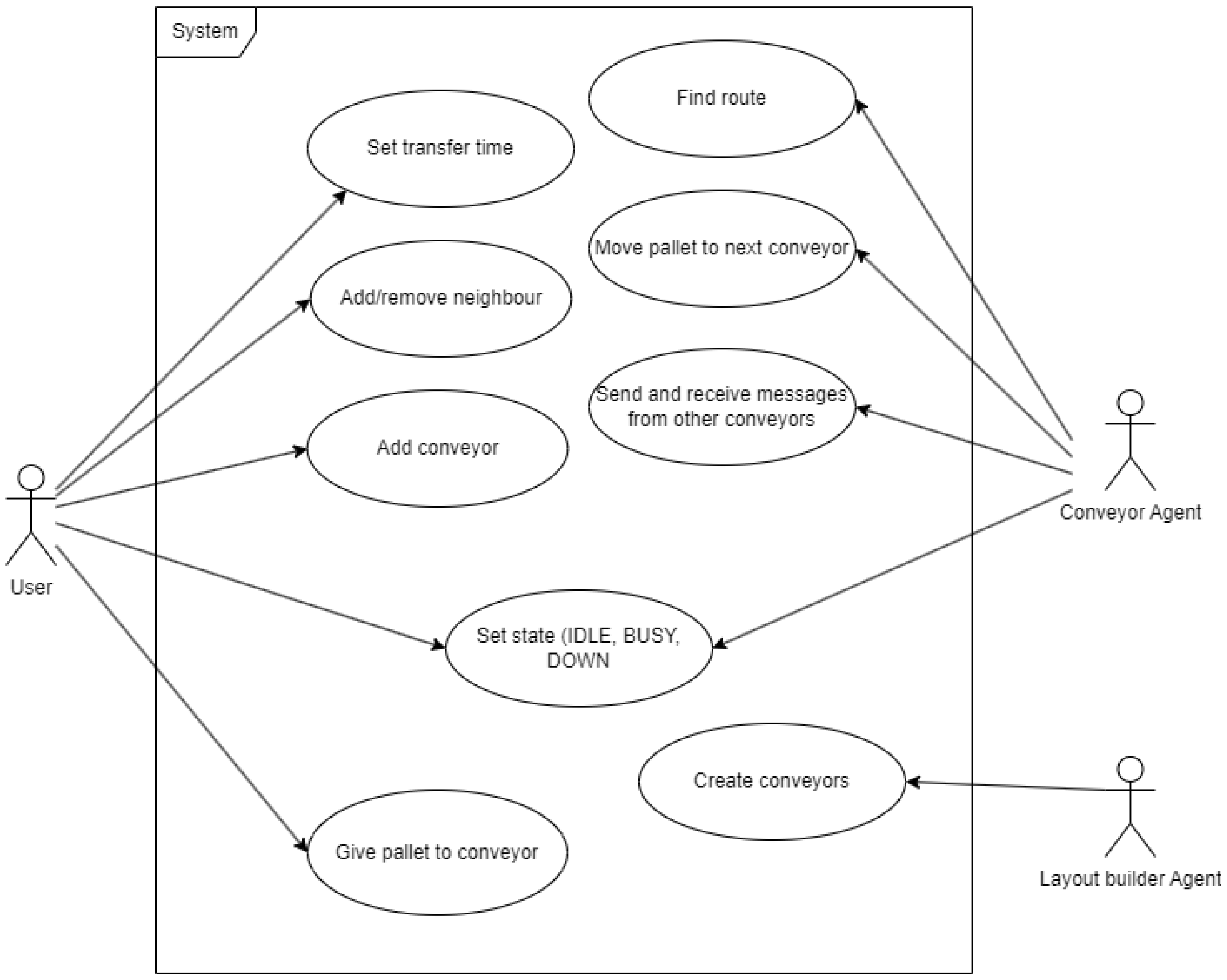


Figure 3: Use case diagram for the conveyor system

Different messages that are sent between conveyors are explained in figure 4.

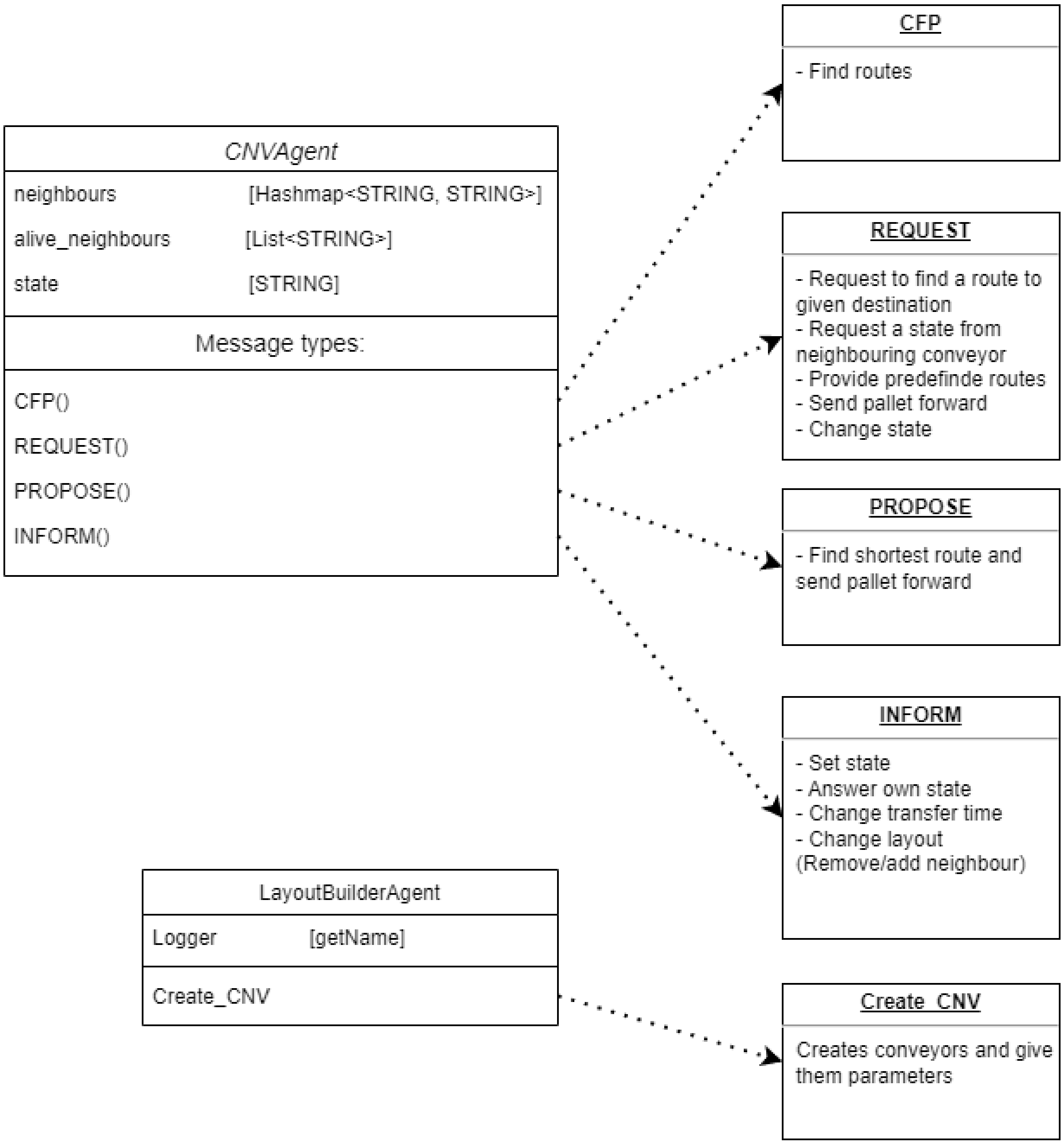

CNVAgent
neighbours [Hashmap<STRING, STRING>]
alive_neighbours [List<STRING>]
state [STRING]
Message types:
CFP()
REQUEST()
PROPOSE()
INFORM()
LayoutBuilderAgent
Logger [getName]
Create_CNV
CFP
- Find routes
REQUEST
- Request to find a route to given destination
- Request a state from neighbouring conveyor
- Provide predefinde routes
- Send pallet forward
- Change state
PROPOSE
- Find shortest route and send pallet forward
INFORM
- Set state
- Answer own state
- Change transfer time
- Change layout (Remove/add neighbour)
Create_CNV
Creates conveyors and give them parameters

Figure 4: Variables and messages of agents

In figure 5 and 6 the basic working of our solution is depicted in a more abstract way. The FIPA communications that happen between the agents are depicted in these. In figure 5 is the basic path finding workings of the agents. The initiating agent checks the state of the neighbouring agents and if the state is idle, it sends a CFP message to the agent to call for a proposal for a possible path. The neighbouring agent then sends back a proposal with the complete path if it is the goal conveyor of the pallet. Else the next conveyor adds itself to the path of the conveyor and repeat the process for its own neighbours. This continues until the message reaches the goal conveyor which then responds to the original initiating conveyor with the complete path.

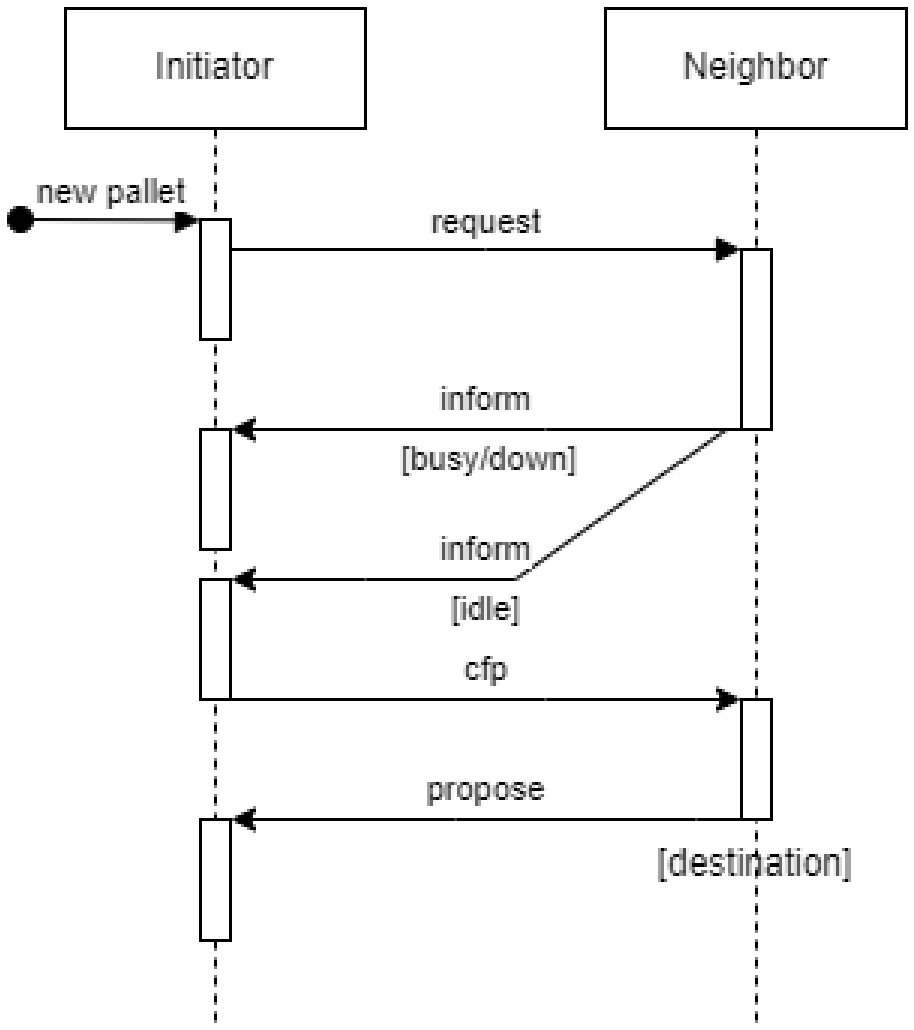


Figure 5: Path finding FIPA-protocol of our system.

In figure 6 is the basic workings for transferring pallets. The previous conveyor checks the state of the next conveyor in the path, and it will respond with its state. If the state is idle the pallet will be transferred to the next pallet. If the conveyor is busy the system will wait and try again and after the wait time or if the conveyor is down, then try to find another path in the path finding way.

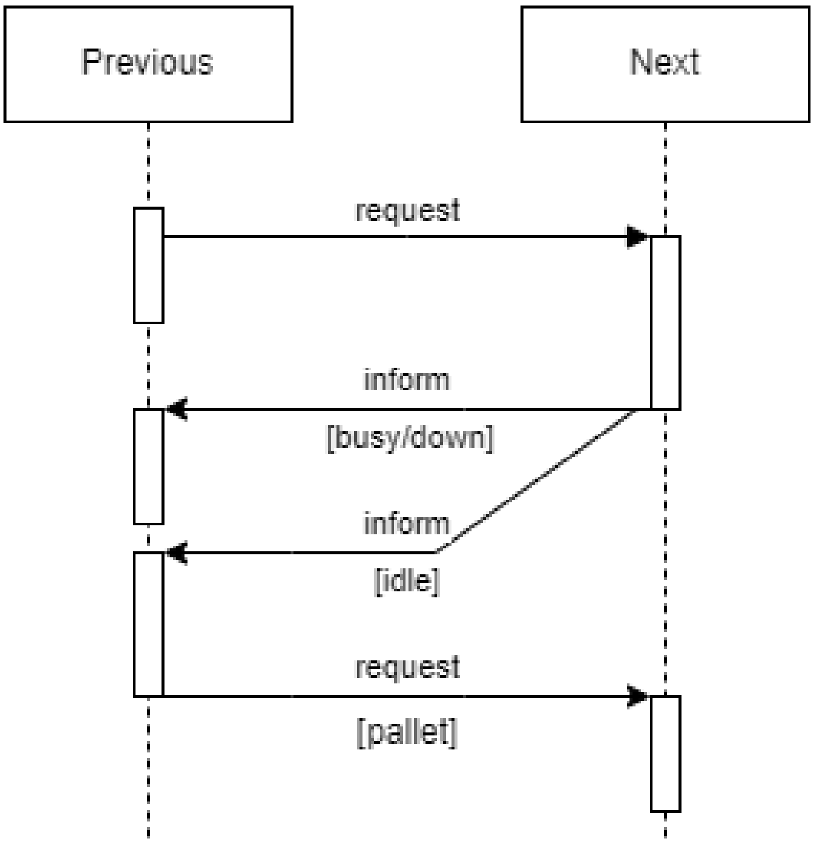


Figure 6: Transfer pallet FIPA-protocol for our system

In figure 7 are the inter-agent communications after the user sent a request message with the following content to the CONV08 agent: {“dest”: “CONV12”}. The source agent asks the state of the next conveyors and if they are idle, a new CFP will be to them. This continues until the destination conveyor is found. When the destination conveyor is found, it sends a proposed message, containing the path, to the source conveyor. After receiving the first propose message the source conveyor waits for 3 seconds and compares all the received path options and chooses the shortest one. Next the source conveyor checks if the next conveyor on the shortest path is idle, and then sends the pallet forward to the next conveyor using a request message.

Again, this continues until the pallet reaches the destination conveyor. If at some point on the path the next conveyor is down or busy, the agent that the pallet is currently on tries to find a new path using the same CFP messages as previously mentioned.

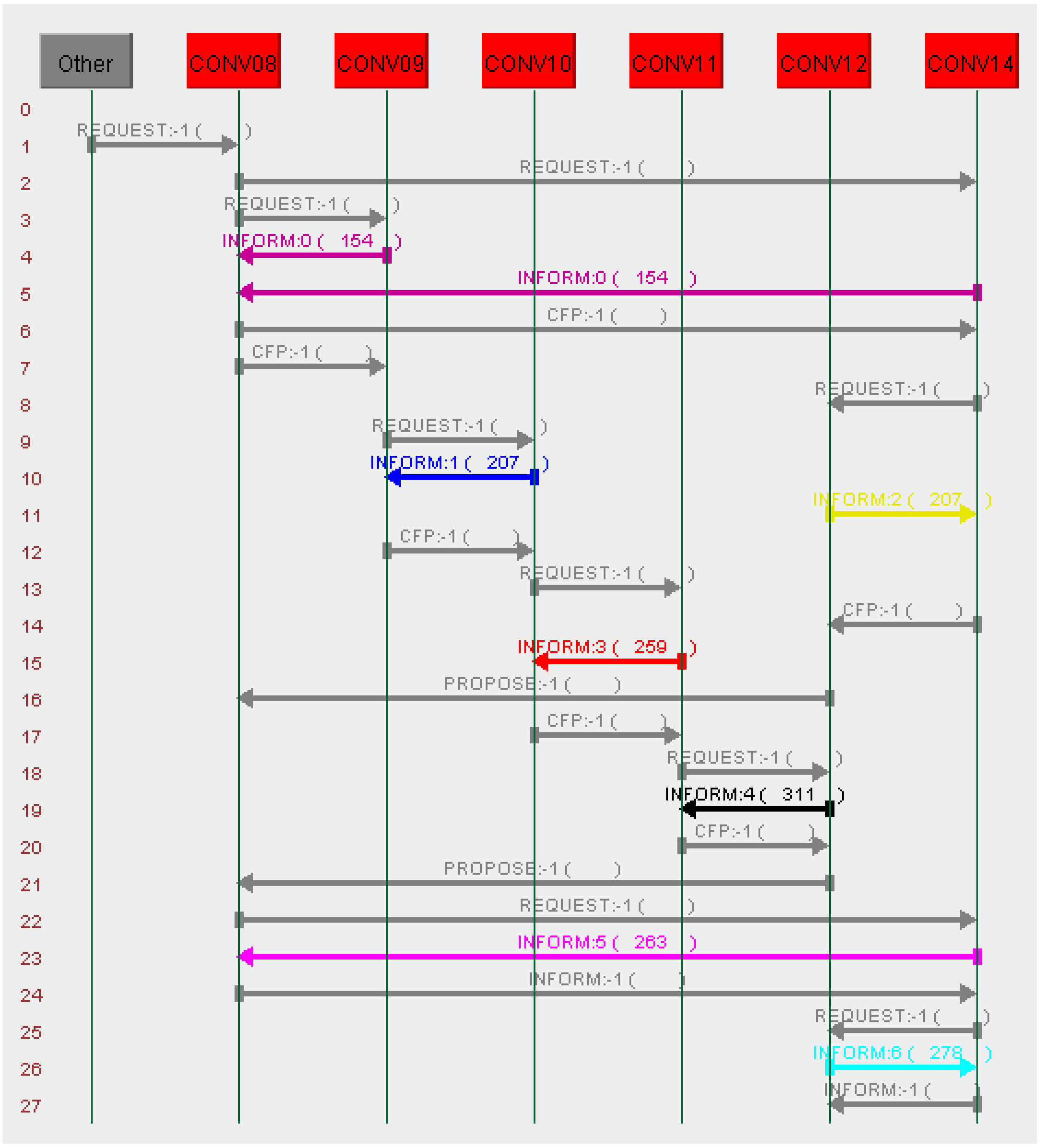


Figure 7: Intra-agent communication after example message.

Weaknesses and strengths of our solution

Weaknesses:

- Transfer pallet function is very convoluted and hard to maintain.
- Takes time to get behind the inner workings of code.
-

Strengths:

- Works as intended in the scenarios we have tested.
- Handles abnormal situations e.g. blockades and broken conveyors
- Fully distributed

Future Work

We are mostly happy with our implementation, and it does fill all the requirements that were set for the assignment. However, there are things that could be added or changed to make the solution better. Here I listed a few ways how our implementation could be improved:

- Make code more readable.
- What happens to the pallets after they have reached their destination?
- To make it easier to configure and reconfigure the conveyor system.

Theoretical background

JADE (Java Agent Development Framework) is used to program distributed agents in java.

Agents are capable of the following traits:

- Autonomy: control over their actions without direct human interaction and have internal sates
- Social ability: interact with each other through a communication language.
- Reactivity: Respondance to changes in environment
- Pro-activeness: Ability to show goal-directed behaviour by taking the iniative

JADE (Java Agent Development Framework)

- Compliant with FIPA
- Distributed agents' platform

FIPA (Foundation for Intelligent Physical Agents)

- Promotes agent-based technology.
- FIPA ACL (FIPA Agent Communication Language)

In JADE there are three behaviour types are “One-shot”, “Cyclic” and “Generic behaviour”. One-shots are executed once and then they die. Cyclic behaviour stays active as long as its agent is alive, and it is called after every event. This is used to handle, for example, message reception. Generic behaviour executes different operations based on the status value. They complete when a given condition is met.

ACL Message consists of these parts:

- Sender
- Receivers
- Communicative act
  - Indicates what is the purpose of the message.
  - REQUEST: sender wants the receiver to perform an action
  - INFORM: sender wants the receiver to be aware of something
  - PROPOSE and CFP (Call for proposal): sender wants to enter into a negotiation.
- Content
- Language

Instructions for testing

How to build layout

The program uses a layout builder agent to build conveyor agents. When created, the layout builder agent creates the test layout show in figure 1. Using the following program arguments the layout builder and the test layout are created automatically when running the program: “-gui -agents Bob:agents.LayoutBuilderAgent”.

Place pallet on conveyor

To place a pallet on a coveyor you can either send the conveyor a request with a destination in the following format: “{"dest": "CONV10"}” or just simply set a conveyors status to busy by sending an inform message to conveyor with the following format: “{“set_state”: “BUSY”}”

modify conveyor transfer time

To modify a conveyors transfer time, you send a Infrom message with the new transfer time x in milliseconds in the following format: “{"transfer_time": "x"}”

modify conveyor status

A conveyors status can be manually modified to be IDLE, BUSY or DOWN. The status can be changed by sending an inform message with the following format: {"set_state": "x"}, where x is one of the three states.

modify layout

To remove a conveyor you can kill the conveyor agent corresponding to the agent you want to eliminate or functionally the same result is achieved also by changing the conveyors state to be DOWN. To add a conveyor a new conveyor needs to be added by creating a new conveyor agent trough JADE. When creating a conveyor agent the agent can be provided with information on which conveyors the can be reached from it. Later you can add more neighbour conveyors by sending the conveyor agent an inform message in the following format: {“neighbor”: “x”}, where x is the neighbour being added.

To remove a connection between two conveyors, send an inform message to the conveyor which should no longer be connected to the next one in the following format: {“remove_neighbor”: “x”}, where x is the neighbour being removed.

provide pre-defined route for pallet

Providing a pre-defined route for a pallet is done by sending a request message to the first conveyor on the route or to a conveyor that has the routes first conveyor in its neighbours. The request message is in the following format: {“route”: [“x”, “y”, “z”]}, where x, y and z are the names of the conveyors on the route.