MAAS Project: Flying Saucers Bakery

Ángela Patricia Enríquez Gómez
Erick Romero Kramer
Ethan Oswald Massey

January 21, 2019

1 Introduction

In this project, the production of a bakery is simulated using JADE, a software framework for developing multi-agent systems [1].

The workflow of the bakery is divided into 5 preparation stages:

- Order processing
- Dough preparation
- Baking
- Packaging
- Delivery

All agents extend a BaseAgent class, which provides the system clock. A production action is allowed every time step.

We have developed and implemented two production stages: Dough preparation and Baking. The interface agent of the Packaging stage has been integrated at the end of our Baking stage to allow testing of the remaining stages. A DummyOrderProcessor is the entry agent of our system. We call it dummy because it does not schedule orders based on the production of each bakery, nor does it compare the prices of the bakeries before placing an order. The DummyOrderProcessor reads the orders (contained in the file clients.json), randomly selects a bakery ID and sends the order to the DoughManager and the BakingInterface of the selected bakery ID.

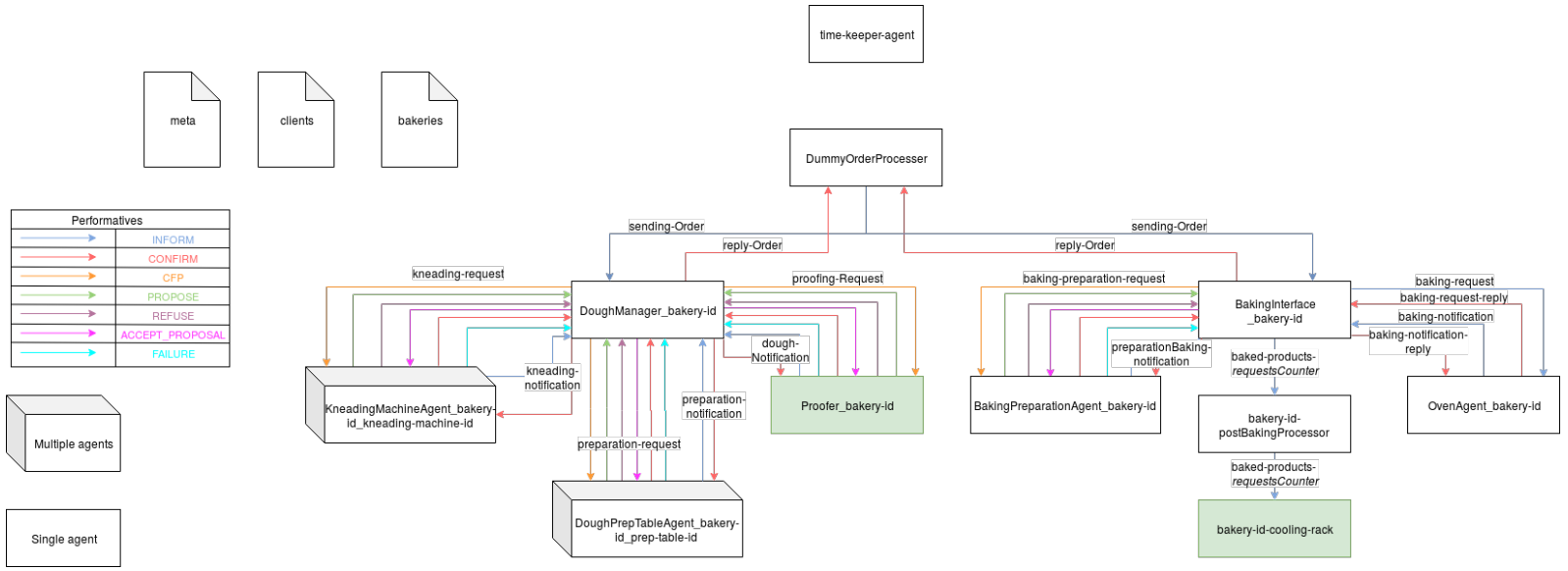

Figure 1: Architecture of the Bakery JADE multi-agent system.

2 System architecture

Table 1 shows the architecture of the Bakery JADE multi-agent system. Each rectangle represents an agent and the interface agents are colored in green. The files bakeries.json contains the bakery information, while the meta.json has the the time step information. The orders are stored in the clients.json files.

3 Agents

Table 1 shows the agents in system, the number of instances of each agent and the method used for creating them. The DummyOrderProcesser, DoughManager agents, Proofer, BakingInterface agents and CoolingRackAgent are created by the Start.java file. Each production stage uses an argument in the Start.java which calls the Initializer of that particular stage. The DougManager creates the agents used in the dough preparation stage, expect for the proofer. Similarly, the BakingInterface creates the agents of the bakery stage, except for the CoolingRackAgent.

Production Stage	Agent	Created by	Number of instances
Order Processing	DummyOrderProcesser	DoughPrepStageInitializer	1
Dough Preparation	DoughManager	DoughPrepStageInitializer	1 per bakery ID
	KneadingMachineAgent	DoughManager	n per bakery ID, where n is taken from bakeries.json
	DoughPrepTableAgent	DoughManger	n per bakery ID, where n is taken from bakeries.json
	Proofer	DoughPrepStageInitializer	1 per bakery ID
Baking	BakingInterface	BakingMasMaasInitializer	1 per bakery ID
	OvenAgent	BakingInterface	1 per bakery ID
	BakingPreparationAgent	BakingInterface	1 per bakery ID
	PostBakingProcessor	BakingInterface	1 per bakery ID
	CoolingRackAgent	BakingStageInitializer	1 per bakery ID

Table 1: Agents in the Bakery JADE.

3.1 Description of the Agents

- **DummyOrderProcesser.** Reads the orders, randomly selects a bakery ID and sends the order to the DoughManager and BakingInterface agents of the selected bakery ID.

- **DoughManager.** Creates the KneadingMachine and DoughPrepTable agents according to the information on the bakery.json. Aggregates orders by product type and maintains queues of products that need kneading, need preparation and need proofing. Coordinates the whole dough preparation stage.

- **KneadingMachineAgent.** Receives CFPs for performing kneading. If the agent is available, it sends a PROPOSE. If the agent is already kneading, it sends a REFUSE.

  The KneadingMachine starts kneading when it receives an ACCEPT_PROPOSAL. The agent can only accept a new *Kneading Request* once it has finished its previous request. When the KneadingMachine agent is done, it sends a *Kneading Notification* to the DoughManager.

- **DoughPrepTableAgent** Receives CFPs for performing dough preparation. If the agent is available, it sends a PROPOSE. If the agent is already preparing products, it sends a REFUSE.

  The DoughPrepTableAgent starts executing the preparation steps when it receives an ACCEPT_PROPOSAL. The agent can only accept a new *Preparation Request* once it has finished its previous request. When the DoughPrepTableAgent agent is done, it sends a *Preparation Notification* to the DoughManager.

- **Proofer.** Receives CFPs for proofing. If the agent is available, it sends a PROPOSE. If the agent is already proofing, it sends a REFUSE.

  The Proofer starts proofing when it receives an ACCEPT_PROPOSAL. The agent can only accept a new *Proofing Request* once it has finished its previous request. When the Proofer agent is done, it sends a *Dough Notification* to the BakingInterface agent. The Proofer is the interface agent between the dough and the baking stages.

- **BakingInterface.** Creates the Oven, the BakingPreparation and the PostBakingProcessor agents. Aggregates orders per product type and maintains queues of products that neeed baking and need preparation. Coordinates the whole baking stage and sends *Cooling Request* messages to the PostBakingProcessor agent of the bakery ID.

- **OvenAgent.** Receives *Baking Request* messages, and replies sending a CONFIRM. It books slots for each baking request. Once a slot has reached the desired temperature, the products are baked. After the baking time is over, the OvenAgent creates and sends a *Baking Notification* containing information of the products that were baked.

- **BakingPreparationAgent.** Receives CFPs for performing baking preparation. If the agent is available, it sends a PROPOSE. If the agent is already preparing products, it sends a REFUSE.

  The BakingPreparationAgent starts executing the preparation steps when it receives an ACCEPT_PROPOSAL. The agent can only accept a new *Baking Preparation Request* once it has finished its previous request. When the BakingPreparationAgent is done, it sends a *Baking Preparation Notification* to the DoughManager.

- **PostBakingProcessor.** Receives a *Cooling Request* message from the BakingInterface and resends it to the CoolingRackAgent. The message sent to the CoolingRackAgent is called *Loading Bay* message but it contains the same information as the *Cooling Request* message.

- **CoolingRackAgent.** Receives *Loading Bay* messages from the PostBakingProcessor agent and performs cooling. It is the interface agent of the next stage.

4 Objects

demo

Object	Parent class	Members type	Members
OrderMas		String	customer_id
		String	guid
		OrderDate	order_date
		DeliveryDate	delivery_date
		Vector<BakedGood>	products
OrderDate		int	day
		int	hour
DeliveryDate		int	day
		int	hour
BakedGood		String	name
		int	amount
		String [ ]	bakedGoodNames
Bakery		String	guid
		String	name
		Point2D	location
		Vector<ProductMas>	products
		Vector<String>	availableProducts
		Vector<Equipment>	equipment
ProductMas		String	guid
		Batch	batch
		Recipe	recipe
		Packaging	packaging
		Double	salesPrice
		Double	productionCost
Bakery		String	guid
		String	name
		Point2D	location
		Vector<ProductMas>	products
		Vector<String>	availableProducts
		Vector<Equipment>	equipment
Equipment		String	guid
		boolean	isAvailable
Batch		int	breadsPerOven
CoolingRequestTuple		String	guid
		int	quantity
		float	coolingDuration
DoughPrepTable	Equipment		
KneadingMachine	Equipment		
Oven	Equipment	int	NUM_SLOTS
		int	coolingRate
		int	heatingRate
		Vector<OvenSlot>	ovenSlots

Table 2: Objects in the Bakery JADE. Part 1.

Object	Parent class	Members type	Members
OvenSlot		float	STARTING_TEMP
		float	currentTemp
		String	ovenGuid
		String	guid
		Integer	quantity
		boolean	available
		boolean	readyToBake
		String	productType
		int	coolingRate
		int	heatingRate
		float	bakingTime
		float	bakingTemp
		int	bakingCounter
ProductStatus		String	guid
		String	Status
		int	amount
		productMas	product
Recipe		int	bakingTemp
		Vector¡Steps¿	steps
Step		String	action
		Float	duration
		String	KNEADING_STEP
		String	ITEM_PREPARATION_STEP
		String	PROOFING_STEP
		String	BAKING_STEP
		String	COOLING_STEP
WorkQueue		LinkedList<ProductStatus>	workQueue

Table 3: Objects in the Bakery JADE. Part 2.

5 Behaviours

Tables 4 and 5 show the behaviors that each agent activates.

Shared Behaviours

- **timeTracker.** All the agents share this behaviour, which checks if the agents are allowed to perform an action. Once the agents are done with an action, they inform the TimeKeeper agent by calling the finished method of the baseAgent. Agents are done with the action of a time step only if they are not currently processing a message. Moreover, the timeTracker behaviour checks if the bakery is in production hours. The production time is from midnight to lunch (00.00 hours to 12 hours).

Production Stage	Agent	Behaviour Name	Behaviour Type
Order Processing	DummyOrderProcesser	timeTracker	CyclicBehaviour
		sendOrder	Behaviour
Dough Preparation	Dough Manager	timeTracker	CyclicBehaviour
		ReceiveOrders	CyclicBehaviour
		checkingKneadingWorkqueue	CyclicBehaviour
		checkingPreparationWorkqueue	CyclicBehaviour
		checkingProofingWorkqueue	CyclicBehaviour
		RequestKneading	Behaviour
		RequestPreparation	Behaviour
		RequestProofing	Behaviour
		ReceiveKneadingNotification	CyclicBehaviour
		ReceivePreparationNotification	CyclicBehaviour
	KneadingMachineAgent	timeTracker	CyclicBehaviour
		ReceiveProposalRequests	CyclicBehaviour
		ReceiveKneadingRequests	CyclicBehaviour
		Kneading	OneShotBehaviour
		SendKneadingNotification	Behaviour
	DoughPrepTableAgent	timeTracker	CyclicBehaviour
		ReceiveProposalRequests	CyclicBehaviour
		ReceivePreparationRequests	CyclicBehaviour
		Preparation	OneShotBehaviour
		SendPreparationNotification	Behaviour
	Proofer	timeTracker	CyclicBehaviour
		ReceiveProposalRequests	CyclicBehaviour
		ReceiveProofingRequests	CyclicBehaviour
		Proofing	OneShotBehaviour
		SendDoughNotification	Behaviour

Table 4: Behaviours in the Bakery JADE. Part 1.

DummyOrderProcesser Behaviours

- **sendOrder.** Sends an *Order* message to the DoughManager and BakingInterface agents of a bakery ID. The order message is sent using the INFORM performative. The behaviour expects an ACL.CONFIRM message from both agents.

Production Stage	Agent	Behaviour Name	Behaviour Type
Baking	BakingInterface	timeTracker	CyclicBehaviour
		ReceiveOrders	CyclicBehaviour
		ReceiveDoughNotification	CyclicBehaviour
		checkingBakingWorkqueue	CyclicBehaviour
		checkingPreparationWorkqueue	CyclicBehaviour
		RequestBaking	Behaviour
		RequestPreparation	Behaviour
		RequestCooling	Behaviour
		ReceiveBakingNotification	CyclicBehaviour
		ReceivePreparationNotification	CyclicBehaviour
	OvenAgent	timeTracker	CyclicBehaviour
		ReceiveBakingRequests	CyclicBehaviour
		checkingBakingRequests	CyclicBehaviour
		Baking	OneShotBehaviour
		SendBakingNotification	Behaviour
	BakingPreparationAgent	timeTracker	CyclicBehaviour
		ReceiveProposalRequests	CyclicBehaviour
		ReceivePreparationRequests	CyclicBehaviour
		Preparation	OneShotBehaviour
		SendPreparationNotification	Behaviour
	PostBakingProcessor	timeTracker	CyclicBehaviour
		ReceiveAndRequestCooling	CyclicBehaviour

Table 5: Behaviours in the Bakery JADE. Part 2.

DoughManager Behaviours

- **ReceiveOrders.** Receives a ACL.INFORM *Order* message from the DummyOrderProcesser and replies with a ACL.CONFIRM message. It adds the orders to a queue of orders.
- **checkingKneadingWorkqueue.** Aggregates the orders that need kneading by product type and creates a *Kneading Request* message. Converts the *Kneading Request* to Json string and activates a RequestKneading behaviour for the *Kneading Request*.
- **checkingPreparationWorkqueue.** Aggregates the orders that need prepara-

tion by product type and creates a *Preparation Request* message. Converts the *Preparation Request* to Json string and activates a RequestPreparation behaviour for the *Preparation Request*.

- **checkingProofingWorkqueue.** Aggregates the orders that need proofing by product type and creates a *Proofing Request* message. Converts the *Proofing Request* to Json string and activates a RequestProofing behaviour for the *Proofing Request*.

- **RequestKneading.** Sends a CFP for a *Kneading Request* to all the KneadingMachine agents. Sends an ACCEPT_PROPOSAL to the first KneadingMachine agent that replied to the CFP with a PROPOSE message and expects an INFORM message as reply. If the CFP fails, i.e., if no KneadingMachine sent a proposal or if the assigned KneadingMachine did no reply with an inform, the products used to create the *Kneading Request* are added back to the needsKneading workqueue.

- **RequestPreparation.** Sends a CFP for a *Preparation Request* to all the DoughPrepTable agents. Sends an ACCEPT_PROPOSAL to the first DoughPrepTable agent that replied to the CFP with a PROPOSE message and expects an INFORM message as reply. If the CFP fails, i.e., if no DoughPrepTable sent a proposal or if the assigned DoughPrepTable did no reply with an inform, the products used to create the *Preparation Request* are added back to the needsPreparation workqueue.

- **RequestProofing.** Sends a CFP for a *Proofing Request* to the Proofer agent. If the Proofer agent replied to the CFP with a PROPOSE message, the DoughManager sends an ACCEPT_PROPOSAL and expects an INFORM message as reply. If the CFP fails, i.e., if the Proofer did not send a proposal or if it did no reply with an inform, the products used to create the *Proofing Request* are added back to the needsProofing workqueue.

- **ReceiveKneadingNotification.** Receives an INFORM *Kneading Notification* message and replies with a CONFIRM. Converts the *Kneading Notification* Json string to an object and adds the kneaded products to the queue needsPreparation.

- **ReceivePreparationNotification.** Receives an INFORM *Preparation Notification* message and replies with a CONFIRM. Converts the *Preparation Notification* Json string to an object and adds the prepared products to the queue needsProofing.

KneadingMachine Behaviours

- **ReceiveProposalRequests.** Receives CFPs for performing kneading. If the KneadingMachine agent is already taken, it replies with a REFUSE and if it is free it sends a PROPOSE.

- **ReceiveKneadingRequests.** Receives ACCEPT_PROPOSALs. If the KneadingMachine agent has already accepted another request it replies with a FAILURE.

If it is still available it replies with an INFORM, converts the *Kneading Request* message to an object, extracts the information needed for kneading and activates the Kneading behaviour.

- **Kneading.** Performs a kneading action if the kneading counter is less than the kneading time of the product being produced. This is a OnceShotBehaviour which is activated at every time step. Once the kneading counter is equal to the kneading time, the KneadingMachine stops kneading, becomes available and activates the SendKneadingNotification behaviour.

- **SendKneadingNotification.** Sends an INFORM message with a *Kneading Notification* to the DoughManager and expects a CONFIRM message as reply.

PreparationTable Behaviours

- **ReceiveProposalRequests.** Receives CFPs for performing preparation. If the DoughPrepTable agent is already taken, it replies with a REFUSE and if it is free it sends a PROPOSE.

- **ReceivePreparationRequests.** Receives ACCEPT_PROPOSALs. If the DoughPrepTable agent has already accepted another request it replies with a FAILURE. If it is still available it replies with an INFORM, converts the *Preparation Request* message to an object, extracts the information needed for performing preparation and activates the Preparation behaviour.

- **Preparation.** The preparation consists of several steps. This OnceShotBehaviour is activated at every time step and performs the preparation actions of each preparation step in a sequential manner. That is, it performs the preparation actions of the first step, then the preparation actions of the second step and so on until reaching the last step. It gets the duration of each preparation step and performs a preparation action each time step. Once the last preparation step has been completed, the PreparationTable stops preparing, becomes available and activates the SendPreparationNotification behaviour.

- **SendPreparationNotification.** Sends an INFORM message with a *preparation-notification* to the DoughManager and expects a CONFIRM message as reply.

Proofer Behaviours

- **ReceiveProposalRequests.** Receives CFPs for performing proofing. If the Proofer agent is already taken, it replies with a REFUSE and if it is free it sends a PROPOSE.

- **ReceiveProofingRequests.** Receives ACCEPT_PROPOSALs. If the Proofer agent has already accepted another request it replies with a FAILURE. If it is still available it replies with an INFORM, converts the *Proofing Request* message to

an object, extracts the information needed for proofing and activates the Proofing behaviour.

- **Proofing.** Performs a proofing action if the proofing counter is less than the proofing time of the product being produced. This is a OnceShotBehaviour which is activated at every time step. Once the proofing counter is equal to the proofing time, the Proofer stops proofing, becomes available and activates the SendDoughNotification behaviour.
- **SendDoughNotification.** Sends an INFORM message with a *Dough Notification* to the BakingInterface agent and expects a CONFIRM message as reply.

BakingManager Behaviours

- **ReceiveOrders.** Receives a ACL.INFORM *Order* message from the DummyOrderProcesser and replies with a ACL.CONFIRM message.
- **ReceiveDoughNotification.** Receives *Dough Notification* messages from the Proofer interface agent and stores the product type, guids and product quantities in a needs baking workqueue.
- **checkingBakingWorkqueue.** Aggregates the orders that need baking by product type and creates a *Baking Request* message. Converts the *Baking Request* to Json string and activates the RequestBaking behaviour for the *Baking Request*.
- **checkingPreparationWorkqueue.** Aggregates the orders that need preparation by product type and creates a *Baking Preparation Request* message. Converts the *Preparation Request* to Json string and activates a RequestPreparation behaviour for the *Preparation Request*.
- **RequestBaking.** Sends an INFORM message containing a *Baking Request* and receives a CONFIRM as reply.
- **RequestPreparation.** Sends a CFP for a *Baking Preparation Request* to all the BakingPreparation agents. Sends an ACCEPT_PROPOSAL to the first BakingPreparation agent that replied to the CFP with a PROPOSE message and expects an INFORM message as reply. If the CFP fails, i.e., if no BakingPreparation sent a proposal or if the assigned BakingPreparation did no reply with an inform, the products used to create the *Baking Preparation Request* are added back to the needsPreparation workqueue.
- **RequestCooling.** Sends *Cooling Request* messages to the PostBakingProcessor agent using an INFORM. Acts as a proxy between the BakingInterface and the CoolingRack agent.
- **ReceiveBakingNotification.** Receives an INFORM *Baking Notification* message and replies with a CONFIRM. Converts the *Baking Notification* Json string to an object and adds the baked products to the queue needsPreparation.

- **ReceivePreparationNotification.** Receives an INFORM *Baking Preparation Notification* message and replies with a CONFIRM. Converts the *Baking Preparation Notification* Json string to an object and creates *Cooling Request* messages for the products that are done with the preparation stage. It activates a RequestCooling behaviour per *Cooling Request*.

OvenAgent Behaviours

- **ReceiveBakingRequest.** Receives and INFORM *Baking Request* message and replies with a CONFIRM. Adds the received *Baking Request* to a list of baking requests.

- **checkingBakingRequests.** Gets the first element in the list of baking requests and books the slots needed for baking the products of each order guid. If not all products of a guid could be assigned to a slot, the baking request is modified with the remaining products and added back to the list of baking requests.

- **Baking.** It heats, cools down the booked slots and bakes. If the slots have the baking temperature it bakes the products for the baking time specified in the recipes. This is a OnceShotBehaviour which is activated at every time step. Once the baking counter is equal to the baking time, the Oven stops baking, releases the slot and and activates the SendBakingNotification behaviour.

- **SendBakingNotification.** Sends a *Baking Notification* of the products that have been baked and expects a CONFIRM.

BakingPreparationAgent Behaviours

- **ReceiveProposalRequests.** Receives CFPs for performing preparation. If the BakingPreparationAgent agent is already taken, it replies with a REFUSE and if it is free it sends a PROPOSE.

- **ReceivePreparationRequests.** Receives ACCEPT_PROPOSALs. If the BakingPreparationAgent agent has already accepted another request it replies with a FAILURE. If it is still available it replies with an INFORM, converts the *Baking Preparation Request* message to an object, extracts the information needed for performing preparation and activates the Preparation behaviour.

- **Preparation.** The preparation consists of several steps. This OnceShotBehaviour is activated at every time step and performs the preparation actions of each preparation step in a sequential manner. That is, it performs the preparation actions of the first step, then the preparation actions of the second step and so on until reaching the last step. It gets the duration of each preparation step and performs a preparation action each time step. Once the last preparation step has been completed, the BakingPreparationAgent stops preparing, becomes available and activates the SendPreparationNotification behaviour.

- **SendPreparationNotification.** Sends an INFORM message with a *Baking Preparation Notification* to the BakingInterface and expects a CONFIRM message as reply.

BakingPreparationAgent Behaviours

- **ReceiveAndRequestCooling.** Receives a *Cooling Request* message from the BakingInterface and sends it to the Cooling Rack agent.

6 Messages

Table 6 shows the messages that the Bakery JADE sends for the order processing, dough and baking stages. Information about the performative, sender, receive and conversation ID is included in the table. All messages are sent as Json strings. In this section we present an example of each type of message.

6.1 Order Processing Stage

Order

```
{
  "customer_id":"customer-001",
  "guid":"order-001",
  "order_date":{
    "day":1,
    "hour":4
  },
  "delivery_date":{
    "day":2,
    "hour":18
  },
  "products":[
    {
      "name":"Bagel",
      "amount":4
    },
    {
      "name":"Donut",
      "amount":5
    },
    {
      "name":"Berliner",
      "amount":3
    },
    {
      "name":"Muffin",
      "amount":2
    },
    {
      "name":"Bread",
```

```
        "amount":1
      }
    ]
}
```

6.2 Dough Stage

Kneading Request

```
{
  "kneadingTime":8.0,
  "guids":[
    "order-001",
    "order-002"
  ],
  "productType":"Bagel"
}
```

Kneading Notification

```
{
  "guids":[
    "order-001",
    "order-002"
  ],
  "productType":"Bagel"
}
```

Preparation Request

```
{
  "productQuantities":[
    4,
    2
  ],
  "steps":[
    {
      "action":"resting",
      "duration":7.0
    },
    {
      "action":"sheeting",
      "duration":6.0
    },
    {
      "action":"twisting",
      "duration":14.0
    },
    {
      "action":"item preparation",
      "duration":5.0
    },
    {
      "action":"filling",
```

```
        "duration":14.0
      }
  ],
  "guids":[
    "order-001",
    "order-002"
  ],
  "productType":"Bagel"
}
```

Preparation Notification

```
{
  "productQuantities":[
    4,
    2
  ],
  "guids":[
    "order-001",
    "order-002"
  ],
  "productType":"Bagel"
}
```

Proofing Request

```
{
  "proofingTime":6.0,
  "productQuantities":[
    4,
    2
  ],
  "guids":[
    "order-001",
    "order-002"
  ],
  "productType":"Bagel"
}
```

6.3 BakingStage

Dough Notification

```
{
  "productQuantities":[
    1,
    2
  ],
  "guids":[
    "order-001",
    "order-002"
  ],
  "productType":"Bagel"
}
```

Baking Request


```
{
  "bakingTemp":200,
  "bakingTime":9.0,
  "productQuantities":[
    4,
    2
  ],
  "slotsNeeded":[
    1,
    1
  ],
  "productPerSlot":4,
  "guids":[
    "order-001",
    "order-002"
  ],
  "productType":"Bagel"
}
```


Baking Notification


```
{
  "productQuantities":[
    4
  ],
  "guids":[
    "order-001"
  ],
  "productType":"Bagel"
}
```


Baking Preparation Request


```
{
  "productQuantities":[
    4
  ],
  "steps":[
    {
      "action":"twisting",
      "duration":14.0
    },
    {
      "action":"frying",
      "duration":10.0
    }
  ],
  "guids":[
    "order-001"
  ],
  "productType":"Bagel"
}
```

Baking Preparation Notification


```
{
  "productQuantities":[
    4
  ],
  "guids":[
    "order-001"
  ],
  "productType":"Bagel"
}
```


Cooling Request = Loading Bay Message


```
[
  {
    "guid":"Bagel",
    "quantity":4,
    "coolingDuration":4.0
  }
]
```

Production Stage	Message	Performative	Sender	Receiver	Conversation ID
Order Processing	Order	INFORM	DummyOrderProcesser	DoughManager and BakingInterface agents of a bakeryID	sending-Order
Dough Preparation	Kneading Request	CFP	DoughManager of bakery ID.	All KneadingMachine agents of bakery ID.	kneading-request
	Kneading Notification	INFORM	KneadingMachine of bakery ID that accepted the kneading request.	DoughManager of bakery ID.	kneading-notification
	Preparation Request	CFP	DoughManager of bakery ID.	All PreparationTable agents of bakery ID.	preparation-request
	Preparation Notification	INFORM	PreparationTable of bakery ID that accepted the preparation request	DoughManager of bakery ID.	preparation-notification
	Proofing Request	CFP	DoughManager of bakery ID	Proofer of bakery ID.	proofing-Request
Baking	Dough Notification	INFORM	Proofer of bakery ID.	BakingInterface of bakery ID	dough-Notification
	Baking Request	INFORM	BakingInterface of bakery ID	Oven agent of bakery ID	baking-request
	Baking Notification	INFORM	Oven agent of bakery ID	BakingInterface of bakery ID	baking-notification
	Baking Preparation Request	CFP	BakingInterface of bakery ID	BakingPreparationTable agent of bakery ID	baking-preparation-request
	Baking Preparation Notification	INFORM	BakingPreparationTable agent of bakery ID	BakingPreparationTable agent of bakery ID	preparationBaking-notification
	Cooling Request	INFORM	BakingInterface of bakery ID	PostBakingProcessor of bakery ID	baked-products + counter
	Loading Bay Message	INFORM	PostBakingProcessor of bakery ID	Cooling Rack of bakery ID	baked-products + counter

Table 6: Messages sent and received in the Bakery JADE.

7 Instructions for running the project

7.1 Run preparation stages in one computer

Run only the Dough Preparation Stage

```
gradle run --args='-doughPrep -scenarioDirectory nameScenarioDirectory'

Example:

gradle run --args='-doughPrep -scenarioDirectory small'
```

Run only the Dough Preparation Stage with visualization

```
gradle run --args='-doughPrepVisual -scenarioDirectory
    nameScenarioDirectory'

Example:

gradle run --args='-doughPrepVisual -scenarioDirectory small'
```

Run both Dough Preparation and Baking Stage

```
gradle run --args='-doughPrep -bakingMasMaas -baking -scenarioDirectory
     nameScenarioDirectory'

Example:

gradle run --args='-doughPrep -bakingMasMaas -baking -scenarioDirectory
     small'
```

7.2 Run preparation Dough and Baking Stages in different computers

- Connect to the same network
- Find the ip address of the server/host machine
- Use port 5555

Run the Baking Stage in the server/host machine

```
gradle run --args="-isHost 192.168.88.182 -localPort 5555
    -bakingMasMaas -scenarioDirectory nameScenarioDirectory -noTK"

Example:

gradle run --args="-isHost 192.168.88.182 -localPort 5555
    -bakingMasMaas -scenarioDirectory small -noTK"
```

Run the doughStage in the client machine

```
gradle run --args="-host 192.168.88.182 -port 5555 -doughPrep
    -scenarioDirectory nameScenarioDirectory"

Example:

gradle run --args="-host 192.168.88.182 -port 5555 -doughPrep
    -scenarioDirectory small"
```

Visualization

The visualization section of the project is meant to give insight into historical performance and trends of the bakeries. Information about used for future visualization is obtained by the "LoggerAgent". The LoggerAgent subscribes and listens to communication between all of the other stages. Currently, this is only implemented for the Dough stage. For example, when a kneading machine finishes an order and passes it on to the preparation table, a message must be sent. The message is also captured by the LoggerAgent. When a message is captured, important information such as which machine was being used and at what bakery. This information is tagged with a timestamp and saved to a buffer. The buffer, at the end of every timestep is then written out to a file.
When the entire project reaches the end time, the file where the information was being logged is closed. At this point, a fully functional and correctly formatted CSV exists on the host computer. Before the LoggerAgent yields to the Timekeeper agent for the last time, it parses the CSV file and displays a graph showing the performance. The CSV can be used for debugging or more elaborate data analysis later.

References

[1] Fabio Bellifemine, Via G Reiss Romoli, Giovanni Rimassa, and Agostino Poggi. JADE A FIPA-compliant agent framework.

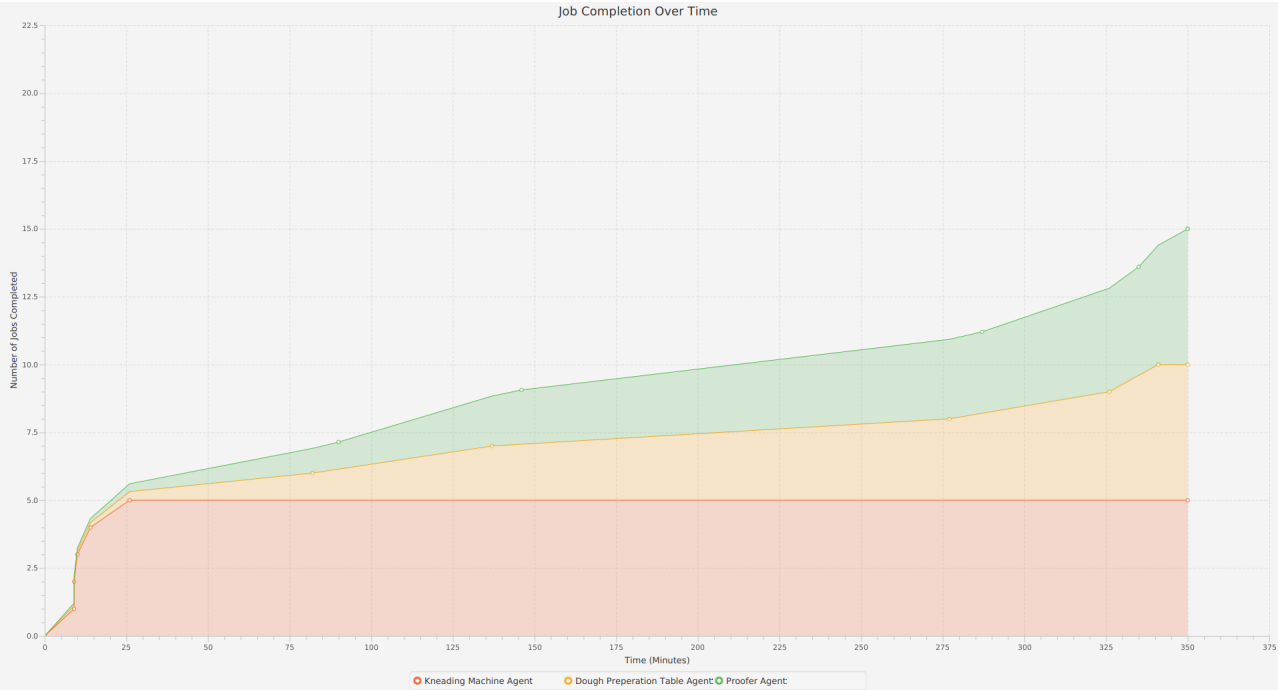

Figure 2: Visualization of the dough stage