**Hochschule**
**Bonn-Rhein-Sieg**
University of Applied Sciences

**Semesterproject**

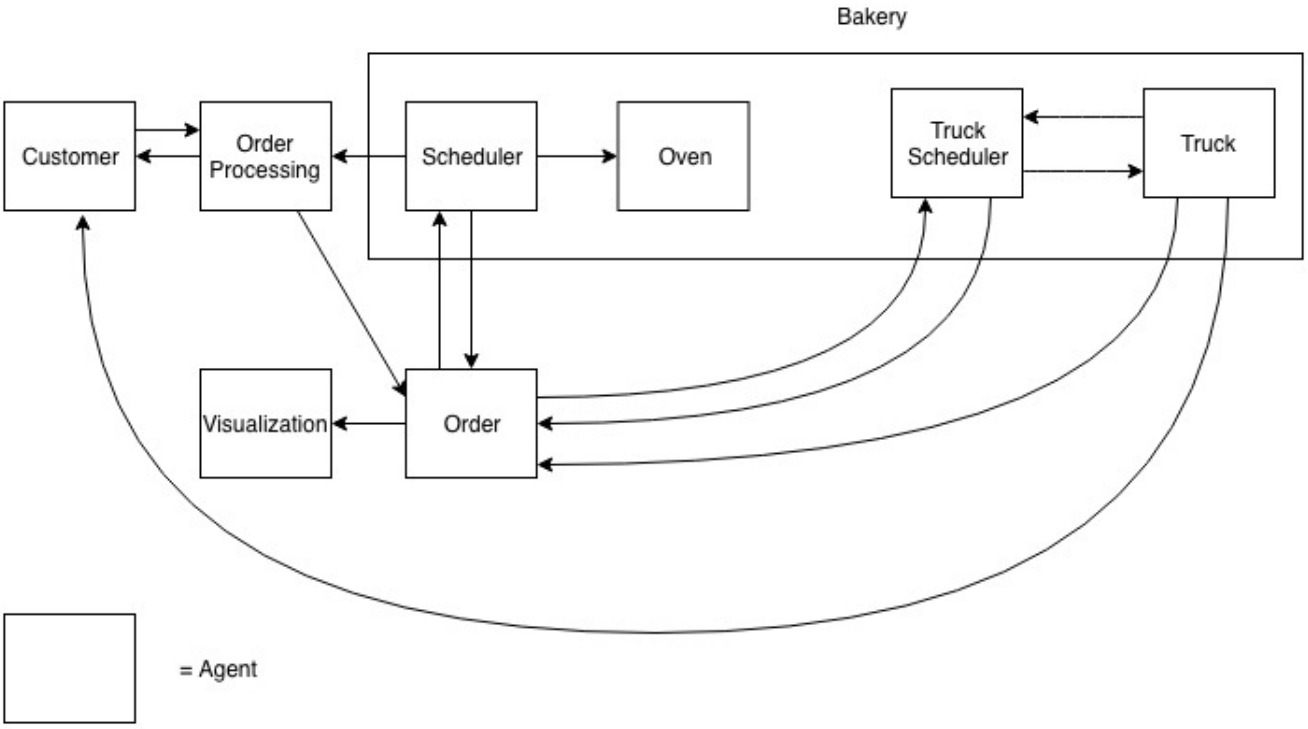
# Multi Agent and Agent Systems

Department of Autonomous Systems
Referent: Prof. Dr. Gerhard Kraetzschmar

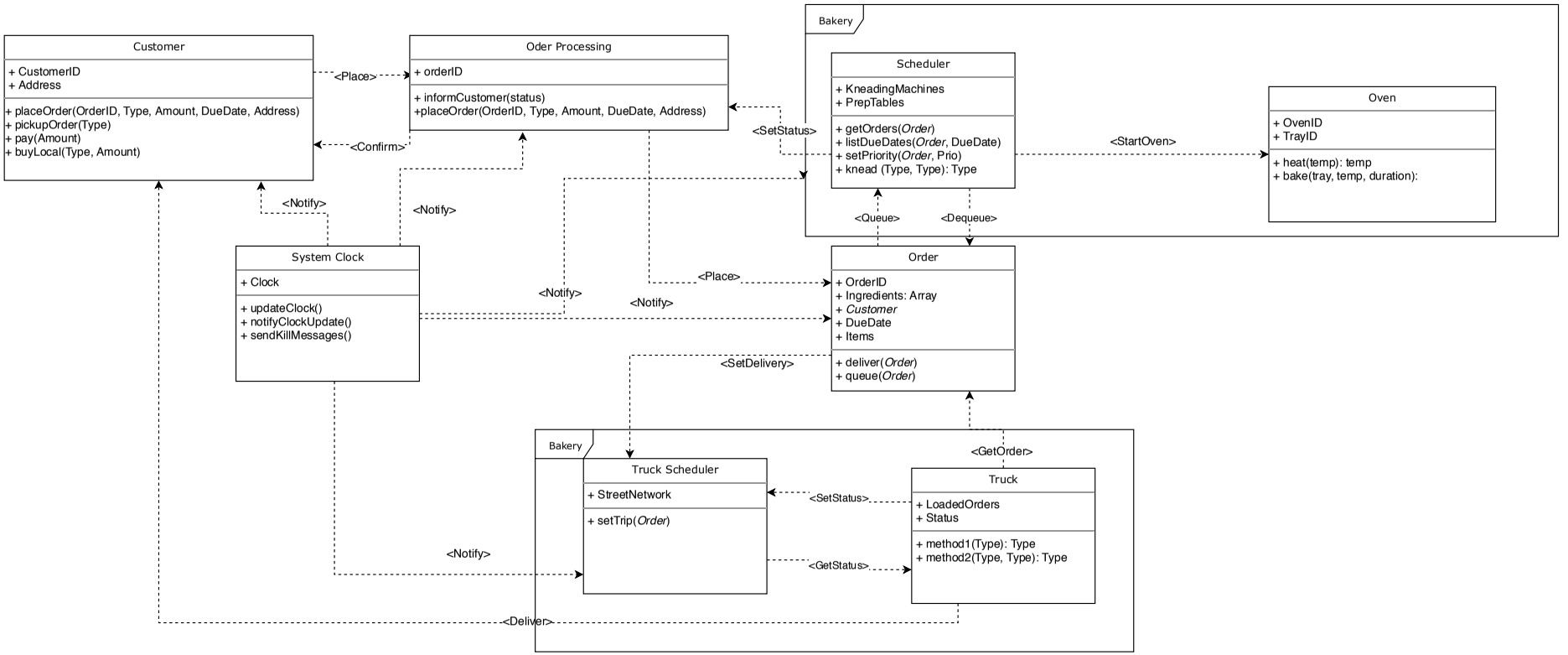
Submitted by: Team PJT

Sankt Augustin 5. November 2018

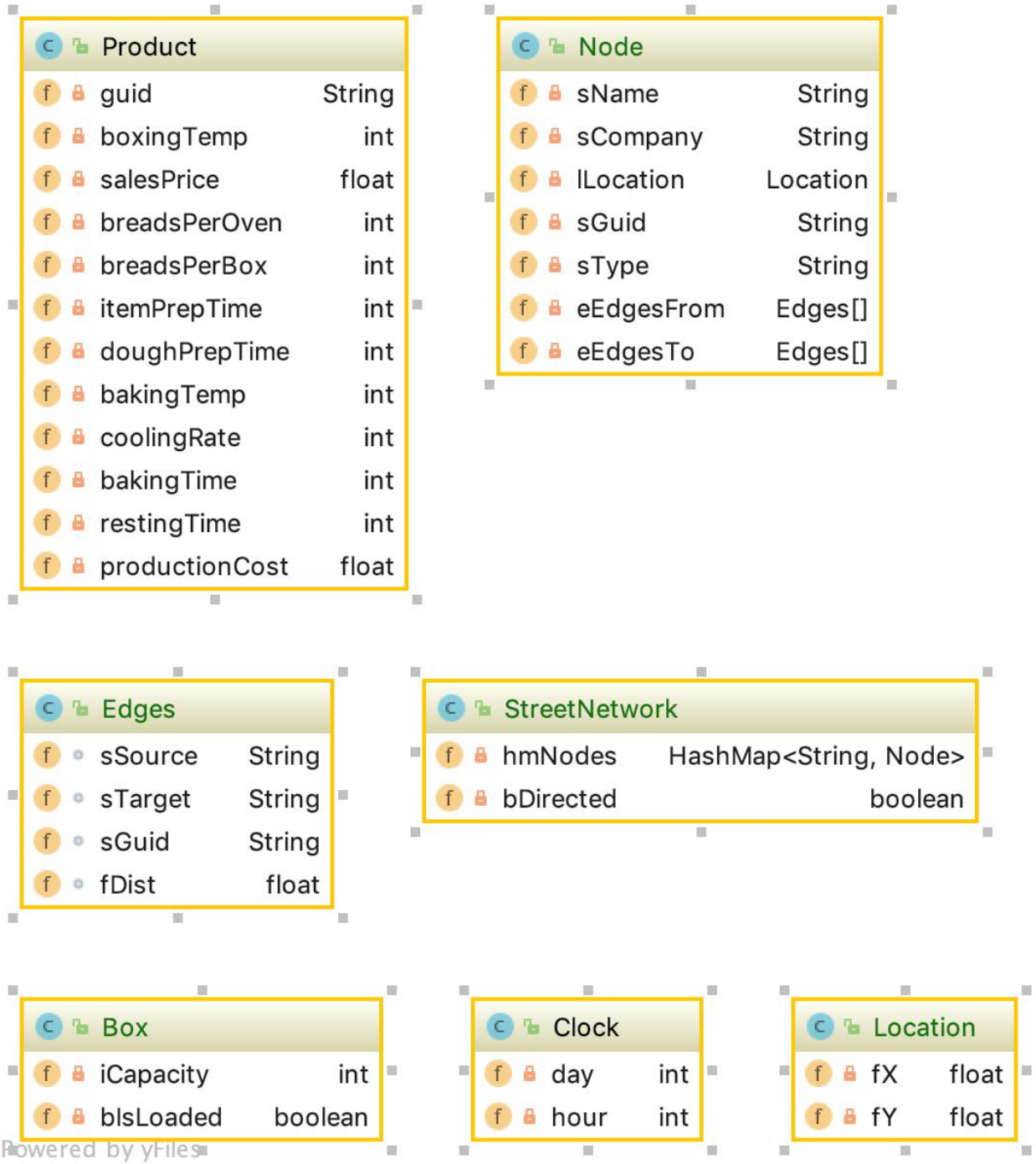
# Architecture



**Agents**


Customer
+ CustomerID
+ Address
+ placeOrder(OrderID, Type, Amount, DueDate, Address)
+ pickupOrder(Type)
+ pay(Amount)
+ buyLocal(Type, Amount)
<Place>
<Confirm>
Oder Processing
+ orderID
+ informCustomer(status)
+placeOrder(OrderID, Type, Amount, DueDate, Address)
Bakery
Scheduler
+ KneadingMachines
+ PrepTables
+ getOrders(Order)
+ listDueDates(Order, DueDate)
+ setPriority(Order, Prio)
+ knead (Type, Type): Type
<SetStatus>
<StartOven>
Oven
+ OvenID
+ TrayID
+ heat(temp): temp
+ bake(tray, temp, duration):
<Queue>
<Dequeue>
<Notify>
<Notify>
System Clock
+ Clock
+ updateClock()
+ notifyClockUpdate()
+ sendKillMessages()
<Notify>
<Notify>
<Place>
Order
+ OrderID
+ Ingredients: Array
+ Customer
+ DueDate
+ Items
+ deliver(Order)
+ queue(Order)
<SetDelivery>
Bakery
<GetOrder>
Truck Scheduler
+ StreetNetwork
+ setTrip(Order)
Truck
+ LoadedOrders
+ Status
+ method1(Type): Type
+ method2(Type, Type): Type
<SetStatus>
<GetStatus>
<Notify>
<Deliver>

# Objects


Product
guid String
boxingTemp int
salesPrice float
breadsPerOven int
breadsPerBox int
itemPrepTime int
doughPrepTime int
bakingTemp int
coolingRate int
bakingTime int
restingTime int
productionCost float
Node
sName String
sCompany String
lLocation Location
sGuid String
sType String
eEdgesFrom Edges[]
eEdgesTo Edges[]
Edges
sSource String
sTarget String
sGuid String
fDist float
StreetNetwork
hmNodes HashMap<String, Node>
bDirected boolean
Box
iCapacity int
bIsLoaded boolean
Clock
day int
hour int
Location
fX float
fY float
Powered by yFiles

# Aggregation of order data

Aggregation of order data can be done in the following manner:

- **An aggregation of a customer's orders for each day or each date <ddd.hh>**
  → It depends which data structure you could use
  - If it is really important to you that you access date by given data format you could use a hashmap. Key is date value is order. Worst performance of searching a hashmap is $O(n) = log(n)$

```
Hashmap<Date, Order> hmMapDaily = new Hashmap<Date, Order>();
hmMapDaily.put(new Date(), new Order());
Order co = hmMapDaily.get(date);
```

  - If it is not that important to use the given dateformat you could use an array. Index is day of a year. That means here worst performance of searching an array given that you know which day you want to search is $O(n) = 1$

- **An aggregation of all orders for a particular product for each day or each date**
  → Hashmap of Hashmaps. One entry within Hashmap represents one product. Key is product value is a hashmap. One Hashmap within Hashmap has as key a date, as value an array of orders.

```
Hashmap<ProductId, Hashmap<Date, Orders[]>> hMapProduct;
hMapProduct.put(new ProductId(), Hashmap<Date, Orders[]>);
Hashmap<Date, Orders[]> hmDate = hMapProduct.get(ProductId);
```

  So hMapProduct would look the following way:

$$hMapProduct = \begin{pmatrix} \{ProductId, Hashmap < Date, Orders[] >\} \\ . \\ . \\ . \\ \{ProductId, Hashmap < Date, Orders[] >\} \end{pmatrix}$$