# Task4

In this task we are asked to model the scenarios using a role-based modeling approach for multi-agent systems, RoMAS.

A role can be seen as a constraint under which an agent takes part in some interactions and behaves in a certain way. In general, an agent can have more than one role at a time, changing dynamically its roles, but in our scenarios we have a one-to-one relationship between roles and agents. If we see our scenarios happening consequently, there is one agent that changes role dynamically, the one that in the Dutch Auction plays the role of a bidder and in the Virtual Tour plays the role of a curator. Since in this task we will try to model the three different scenarios individually, we will never have the schema about the roles’ transitions.

The modelling follows the steps:

1. Capture use cases;
2. Identify roles from use cases;
3. Construct role organization;
4. For each role, if the appropriate agent does not exit, then go to 5.; else
5. Bind roles to agents
6. Describe dynamic properties of bind relation between agents and roles
7. Go to 6.
8. Generate agents according to roles. Go to 4.I.;
9. Generate codes for agents with roles bound.

## Virtual Tour

First of all, to understand the interactions between roles, we propose a role organization overview:

### 1. Use cases

Capturing use cases is useful for outlying the system events and their interactions. We tried to represent the interaction focusing also on the goals of actions.

In the following diagram, for instance, we claim that the curator “receives a request for a virtual tour” and then “gets a virtual tour”, where *gets* stands for *creates*, while the other agents actually *receive* the tour from another agent.



### 2. Roles



### 

### 3. Role Organization



### 4. Agents

In order to provide a good representation of the agents, we show the actual methods and behaviors that implemented agents have. The attributes are the same ones of the roles each agent represents.

|  |
| --- |
| Agent GUIDE |
| <Ability>  showTourGuideInfo()  register()  <Behavior rule>  *TourGuide*:  Subscribe  ReceiveRequest  MakeTourInitiator |

|  |
| --- |
| Agent VISITOR |
| <Ability>  showUserInfo()  findCurator()  findTourGuide()  getTour()  <Behavior rule>  *Profiler*:  RequestArtifactsIDs  RequestArtifactsDetails |

|  |
| --- |
| Agent CURATOR |
| <Ability>  showCuraorInfo()  register()  <Behavior rule>  *Curator*:  RespondToProfilerAgent  RespondToTourGuideAgent |

## Dutch Auction

In this scenario we have two kind of agents that interact: auctioneer and bidders. The role organization overview is:

where n2.

### 1. Use cases



### 2. Roles



### 3. Role Organization



### 4. Agents

|  |
| --- |
| Agent ARTISTMANAGER |
| <Ability>  showArtistManagerInfo()  findBidders()  informStatus()  informNewBid()  <Behavior rule>  *Auctioneer*:  SendNewBid  ReceiveProposals |

|  |
| --- |
| Agent CURATOR1 |
| <Ability>  registerAsBidder()  setCuratorsBid(previousBid)  <Behavior rule>  *Bidder*:  ParticipatingInAuction |

## Intra-Platform Auctions

RoMAS does not provide a particular model for mobile agents. Since it is based on roles, we decided to model the Intra-Platform Auctions scenario using three different roles: the manager (that is the original auctioneer agent), the cloned auctioneer and the bidder. The role organization overview is



### 1. Use Cases



The manager first creates two clones, that will be the auctioneers in two separate containers. Furthermore, the original bidders clone themselves, but since they participate to the auctions with the clones, it is not necessary to consider new roles. Finally, the cloned auctioneers communicate the results to the manager at the end of the auctions.

### 2. Roles





### 3. Role Organization



### 4. Agents

|  |
| --- |
| Agent ORIGINALAGENT |
| <Ability>  registerAsArtistManager()  getLocations()  doClone()  <Behavior rule>  *Manager*:  ReceivingBehavior |

|  |
| --- |
| Agent CLONEDAUCTIONEER1 |
| <Ability>  showArtistManagerInfo()  findBidders()  informStatus()  informNewBid()  getOriginalAgent()  sendMessageAndReturn()  afterClone()  afterMove()  <Behavior rule>  *Auctioneer*:  SendNewBid  ReceiveProposals |

|  |
| --- |
| Agent CURATOR1 |
| <Ability>  registerAsBidder()  setCuratorsBid(previousBid)  afterClone()  <Behavior rule>  *Bidder*:  ParticipatingInAuction |

## Comparisons with GAIA

Both modelling techniques focus on the roles and their description. Indeed, the *Roles Model* step in GAIA and the *Role* step in RoMAS provide almost the same results.

But, using GAIA, we are able to obtain a more complete understanding of the interactions between roles, thanks to the *Interaction Model*. Using this step, the design phase is straight forward. On the other hand, modelling the scenarios via RoMAS requires less details about interactions, because it simply provides a big picture using *Use Cases*.

Finally, m-GAIA provides also the *Mobility Model*, while RoMAS does not, and for this reason we had to adapt the existing steps to model the Intra-Platform Auctions scenarios, even if the results via m-GAIA are more correct and complete.