Project - ID2209
Distributed AI and Intelligent Agent

Weherage, Pradeep Peiris

December 10, 2015

Task 1 - Requirement Statement

Requirement Statement

SmartMuseum is an artificial society (System) in which Profiler Agents, Curator Agents and TourGuide Agents are functioning together.

Profiler Agents in SmartMuseum represents users who interested in artifacts from different Museums/Galleries. Users manage their profile attributes such as age, occupation, gender and interest via Profiler Agent's GUI.

Curator Agents manage artifacts of Gallery or Museum. Each artifact is defined with its attributes such as name, creator, date of creation, place of creation and genre. Curator agents can be requested for more detail of given artifact.

TravelGuide Agent maintain a catalog of artifacts against its genre and other attributes with registered Curator Agents. TravelGuide recommends Virtual Tours upon request from Profiler Agent considering its profile attributes.

Artist Agents in SmartMuseum produce artifacts. And arrange auctions to sell the artifacts. It uses open-cry strategy of Dutch Auction with start price and reserved price. Price is reduced in time basis until it reaches the reserved price. It takes the first bid as the winning price and informed all participant Curator Agents about the status of the auction, otherwise auction will be closed without a winner. Each curator agent plays it own strategy in bidding process. Curators consider the detail of artifact against the demand with its registered profiler agents in defining the bidding strategy for the auction.

Curator agents also perform Dutch Auction with participation of Profiler Agents to sell back any artifacts. Profiler agents may also predefined its strategy in bidding process.

Task 1 - Gaia Model

Gaia is a methodology for designing and modeling of Agent's Structure and Society of Multi Agent System (MAS) development process. It defines the structure of MAS in term of Role Model and interaction protocol between the roles.

Roles Model

Role: Profiler
Description: Profiler role manages user's profile information such as age, occupation, gender and interest, and looking for recommended tours.
Protocols and Activities: RecommendedTours, CallForRecommendedTours, WaitForRecommendedTours, ShowRecommendedTours, UpdateProfile.
Permissions: generates *profileAttributes* *// create Profile attributes* update *profileAttributes* *// change Profile attributes* update *visitedTours* *// update list of visited items* read *recommendedTours* *// read Recommend tours*
Responsibilities: **Liveness:** Profiler = *(RecommendedTours,CallForRecommendedTours.WaitForRecommendedTours.ShowRecommendedTours)*∞ **Safety:** *profileAttributes.interest* is not blank

Role: Curator
Description: Curator role manages artifacts of Gallery/Museum, and respond with detail of artifact upon request for given artifactid/name

Protocols and Activities:
ArtifactDetailRequest, ListenForArtifactDetailRequest, ResponseWithArtifactDetail, CreateArtifactDetail, UpdateArtifactDetail.

Permissions:

generates	*artifactDetail*	*// create artifacts with detail attributes*
update	*artifactDetail*	*// change artifact attributes*
read	*artifactDetail*	*// read artifact attributes*

Responsibilities:
Liveness:
Curator = *(*ArtifactDetailRequest.*ListenForArtifactDetailRequest.ResponseWithArtifactDetail)*∞

Safety:
artifactDetail is not blank

Role: TravelGuide

Description:
TraveGuide role manages registered Curators and maintain a catalog of artifacts from each Curators. It recommends virtual tours for Profilers considering its interest and other attributes.

Protocols and Activities:
RecommendedTours, ListenOnRequestFromProfilers, ListenOnArtifactsFromCurators, UpdateArtifactCatalog, UpdateProfilerRegistry, UpdateCuratorRegistry.

Permissions:

generates	*artifactCatalog*	*// create Artifact catalog*
update	*artifactCatalog*	*// change Artifact catalog*
read	*artifactCatalog*	*// read Artifact catalog*

Responsibilities:
Liveness:
TravelGuide = *(RecommendedTours.ListenOnRequestFromProfilers.RecommendVirtualTours)*∞
TravelGuide = *(ArtifactDetailRequest.ListenOnArtifactsFromCurators.UpdateArtifactCatalog)*∞

Safety:
artifactCatalog is not blank

Role: Artist

Description:
Artist role produces new artifacts or make copies of existing artifacts.

Protocols and Activities:
CreateArtifactDetail, ManageArtifactList.

Permissions:

generates	*artifactList*	*// create list of artifact*
update	*artifactList*	*// change artifact detail*

Responsibilities:
Liveness:
-

Safety:
artifactList is not blank

Role: Auctioneer

Description:
Auctioneer role publishes availability of new artifact with its details and call for an auction.

Protocols and Activities:
PublishAuctionArtifact, DutchAuction, StartDutchAuction, UpdateCurrentPrice, CloseDutchAuction, ShowArtifactDetail, ShowAuctionStatus

Permissions:

generates	*auctionArtifact*	*// create auction artifact*
generates	*startPrice*	*// define start price*
generates	*reservedPrice*	*// define reserved price*
update	*currentPrice*	*// update Current price*

Responsibilities:
Liveness:
Auctioneer =
(PublishAuctionArtifact.StartDutchAuction.ListenForBids.UpdateCurrentPrice.CloseDuctchAuction.PublishAuctionStatus)∞

Safety:
currentPrice > reservedPrice

Role: AuctionParticipent

Description:
Auction Participant role register on called auction and call bids on given strategy.

Protocols and Activities:
DutchAuction, ListenOnBids, CallForBid

Permissions:

update	*bidPrice*	*// update current bid price*
read	*bidStrategy*	*// read bidding strategy*

Responsibilities:
Liveness:
AuctionParticipent = *(RegisterInAuction.ListenOnBiddingPrice.CallForBid)∞*

Safety:
bidPrice doesn't vialote bidStrategy

Interaction Model

Interaction model represents the relationship between roles in a multi agent system.

RecommendedTours		*(Input/Output)*
Profiler	TravelGuide	Profile attributes
Request for recommended tours		Recommended Tours

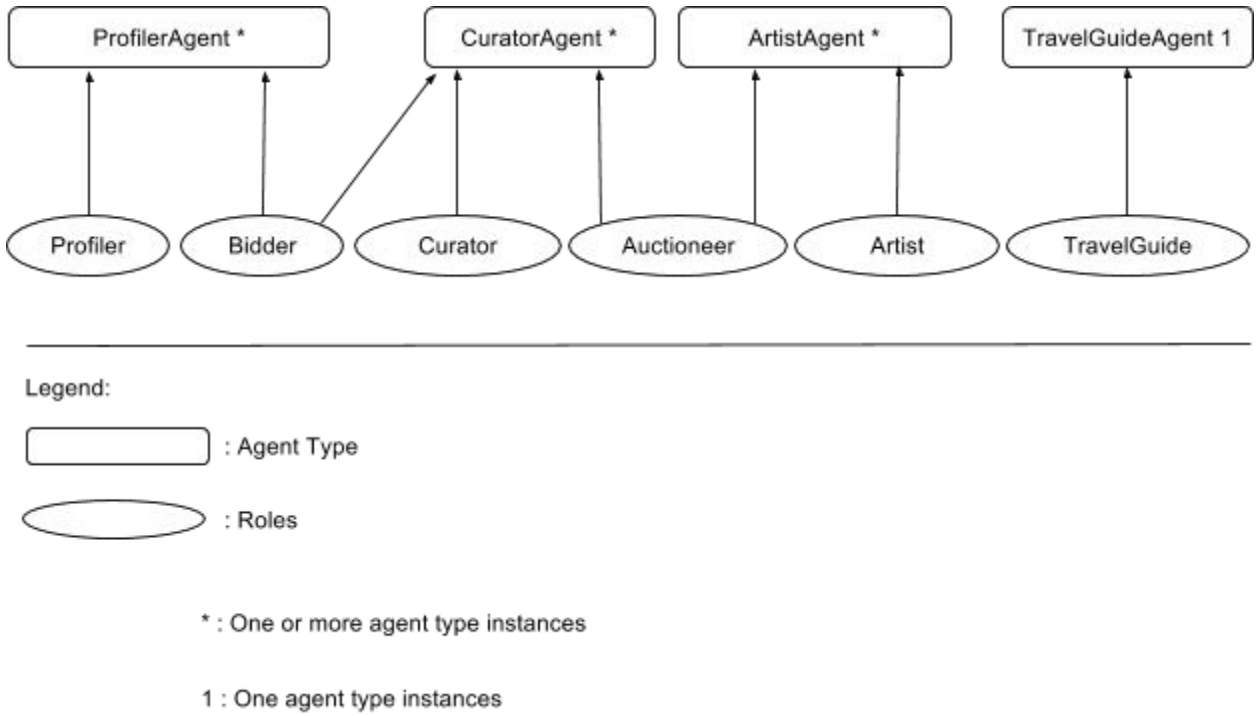
ArtifactDetailRequest		*(Input/Output)*
Curator	Profiler, TravelGuide	Artifact id/name
Request for artifact detail		Artifact Detail

PublishAuctionArtifact		*(Input/Output)*
Auctioneer	Participants	Artifact detail
Publish detail of auction artifact item		Start Price

DutchAuction		*(Input/Output)*
Auctioneer	Participants	Start Price
Auctioneer publish with start price and reduce price according to internal strategy until it reaches reserved price or accepted bid from a participant.		Auction Status

Agent Model

Agent Model identifies different agent types in a multi agent system.



Service Model

Service model represents services with their properties associated with Agent Roles. A service is a single coherent block of activity in which an agent will enage.

Role	**Profiler**
Service	UpdateProfile
Inputs	name, age, occupation, gender and interest
Outputs	View of the user profile
Pre-condition	name and interest are mandatory
Post-condition	accept user profile

Role	**Profiler**
Service	ShowRecommendedTours
Inputs	List of recommended tours
Outputs	View of recommended tours
Pre-condition	-
Post-condition	Valid item of recommended tours

Role	**Curator**
Service	CreateArtifactDetail, UpdateArtifactDetail
Inputs	name, creator, date of creation, place of creation and genre
Outputs	Updated of artifact detail
Pre-condition	name, creator and genre are mandatory
Post-condition	accept updated artifact detail

Role	**TravelGuide**
Service	UpdateArtifactCatalog
Inputs	id and genre
Outputs	Updated of artifact catalog
Pre-condition	genre is mandatory
Post-condition	accept updated artifact catalog

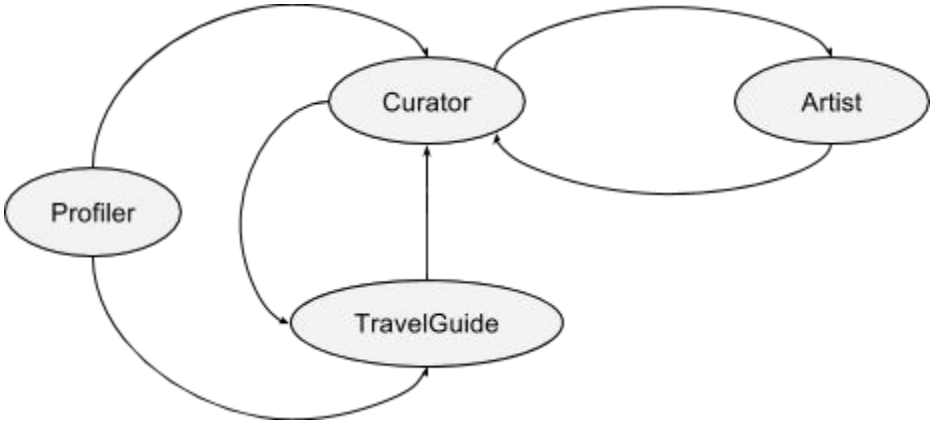
Role	**Artist**
Service	CreateArtifactDetail
Inputs	name, date of creation, place of creation and genre
Outputs	artifact detail
Pre-condition	name and genre are mandatory
Post-condition	accept updated artifact detail

Role	**Auctioneer**
Service	ShowAuctionStatus
Inputs	-
Outputs	Status view of Auction
Pre-condition	active auction
Post-condition	-

Acquaintance Model

Acquaintance Model defines the communication links between different agent type. Typically it can be represented as a graph in which nodes represent agent types and arcs represent the communication pathways.

	Profiler	**Curator**	**TravelGuide**	**Artist**	**Auctioneer**
Profiler		*I*	*I, A*		*I*
Curator				*I,A*	*I*
TravelGuide	*I*	*I, A*			
Artist		*I,A*			
Auctioneer		*A*		*A*	
Legend: I : Interact A: Acquaintances					



Mobility Model

Place Types

Place Types	Description	Instances
Auction House	Container where auction take place	+
Artist Home	Container that Artist agent lives/operate	+
Curator Home	Container for Curator agents to operate. This can be a Gallery or a Museum	+
TravelGuide Home	Container for TravelGuide agent to operate.	1
Profiler Home	Container for Profiler agent to operate.	*

Agents and Places Specification

Agent Type	Mobile	Place Types	Constraints
Artist	Yes	Artist Home, Auction House	
Curator	Yes	Curator Home Auction House	
TravelGuide	No	TravelGuide Home	
Profiler	Yes	Profiler Home Auction House Curator Home	

Cardinality of Agents and Places

The cardinality between agent types and place types shows how many agents of an agent type can reside in a place of a place type.

Artist (1) -----> Artist Home (1)
Artist (+) -----> Auction House (+)
Curator (1) -----> Curator Home (1)
Curator (2..*) -----> Auction House (1)
TraveGuide (1) -----> TravelGuide Home (1)
Profiler (1) -----> Profiler Home (1)
Profiler (2..*) -----> Auction House (1)
Profiler (*) -----> Curator Home (1)

Travel Schema of Mobile Agent Types

The travel schema of each mobile agent type includes origin, final destination, list of atomic movements, and paths. The origin is the place type where the mobile agent starts the movement and the final destination is the place type where mobile agent will reside after it completed the tasks assigned. The atomic movement represents the smallest granularity movement of agents. And paths consists a list of atomic movements that the mobile agent may travel.

Agent Type: Artist Agent

Description:
Artist produces new artifacts or make copies of existing artifacts.

Origin:
Artist Home

Final Destination:
Artist Home

List of Atomic Movements:
ArtistHome_To_AuctionHouse
Artist or clone of Artist is moved to AuctionHouse to run an auction

AuctionHouse_To_ArtistHome
Artist is moved back to ArtistHome at completion of auction

Paths:
Run_Auction: ArtistHome_To_AuctionHouse, AuctionHouse_To_ArtistHome

Agent Type: Curator Agent

Description:
Curator manages artifacts of Gallery/Museum, and respond with detail of artifact upon request for given artifactid/name

Origin:
Curator Home

Final Destination:
Curator Home

List of Atomic Movements:
CuratorHome_To_AuctionHouse
Curator is moved to AuctionHouse to participate in an auction

AuctionHouse_To_CuratorHome
Curator is moved back to CuratorHome at completion of auction

Paths:
Participate_Auction: CuratorHome_To_AuctionHouse, AuctionHouse_To_CuratorHome

Agent Type: Profiler Agent

Description:
Profiler role manages user's profile information such as age, occupation, gender and interest, and looking for recommended tours.

Origin:
Profiler Home

Final Destination:
Profiler Home

List of Atomic Movements:
ProfilerHome_To_AuctionHouse
Profiler is moved to AuctionHouse to participate in an auction

AuctionHouse_To_ProfilerHome
Profiler is moved back to ProfilerHome at completion of auction

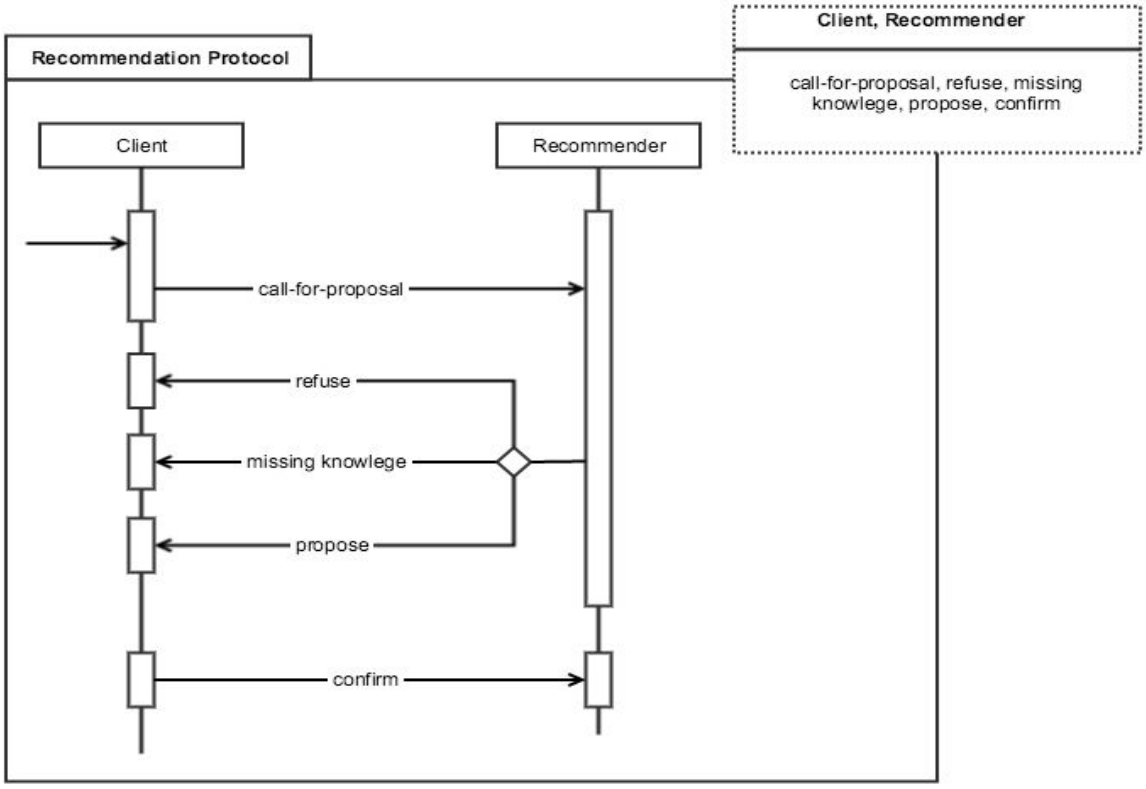
ProfilerHome_To_CuratorHome
Profiler is moved to CuratorHouse to visit artifacts of the Curator

CuratorHome_To_ProfilerHome
Profiler is get back to ProfilerHome from CuratorHome

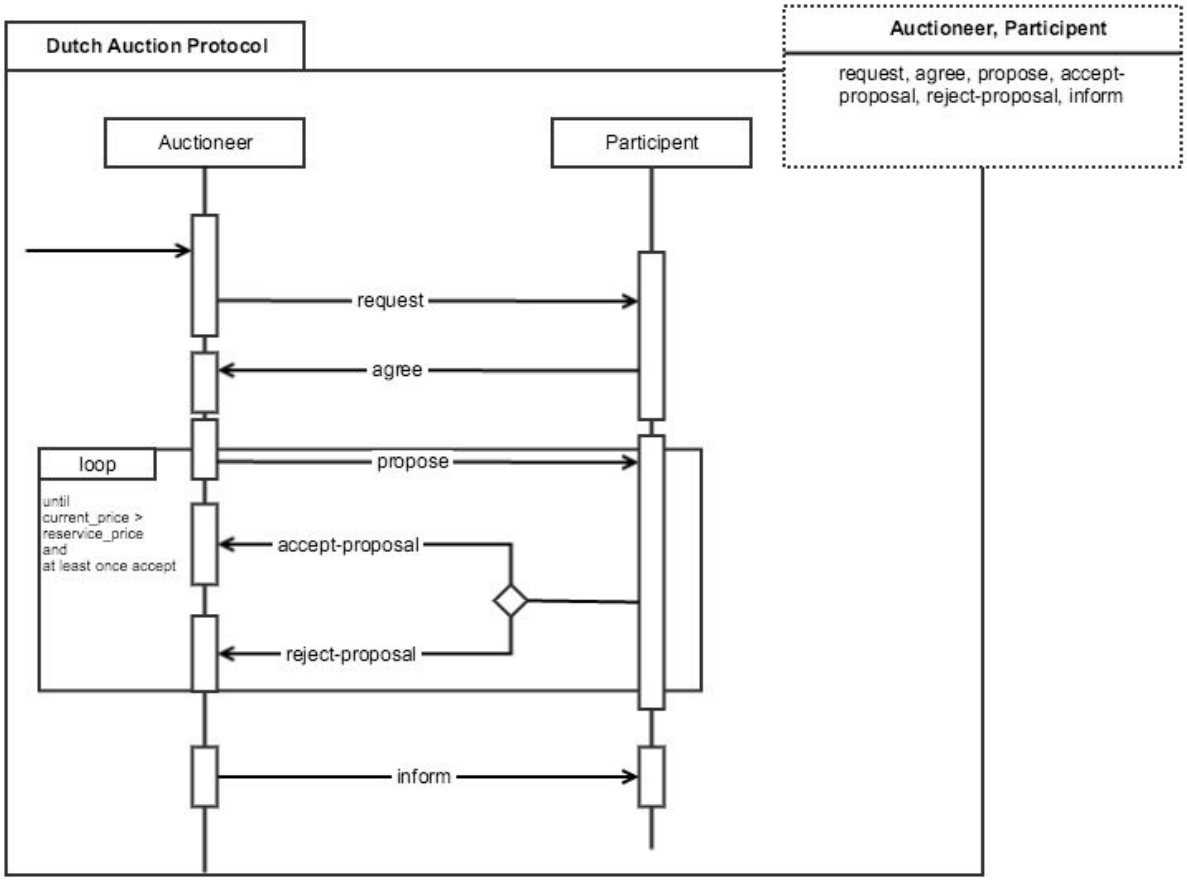
Paths:
Participate_Auction: ProfilerHome_To_AuctionHouse, AuctionHouse_To_ProfilerHome
Visit_Artifact: ProfilerHome_To_CuratorHome, CuratorHome_To_ProfilerHome

Task 2 - Model Interaction in AgentUML

Overall Protocol



The above diagram shows the Recommendation Protocol as a reusable module of processing that can be applied agents communication. The protocol is initiated with a Client and call for proposal of recommendation request to the Recommender. The Recommender use its knowledge of the requester (profile information) and knowledge of the request in replying back to the client. The reply would be a propose of Recommendation list or it would even refuse the request.
The above protocol is applied in communication in between Profiler and TravelGuide agents.

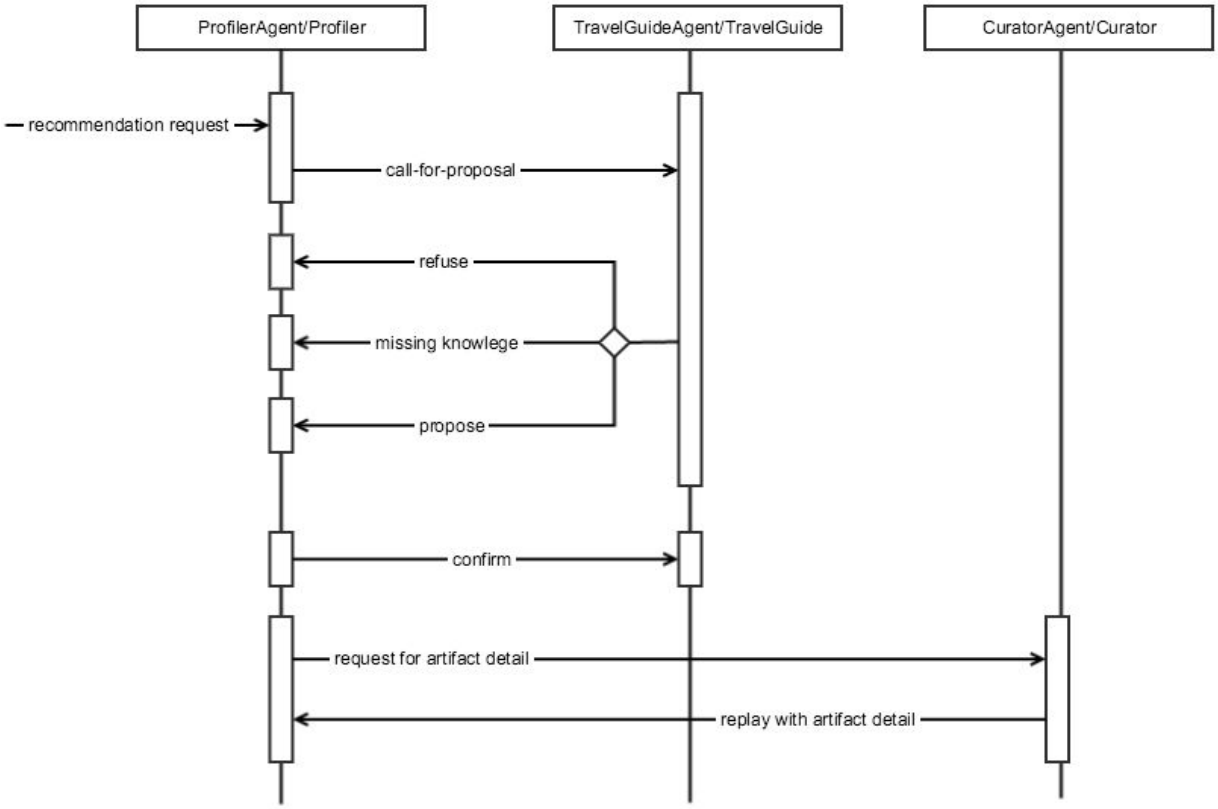


The above diagram shows the Dutch Auction Protocol as a reusable module of executing a Dutch Auction in between agents. The Auctioneer initiate the protocol requesting participants to take part in the auction. The request contains required information of the auction item.

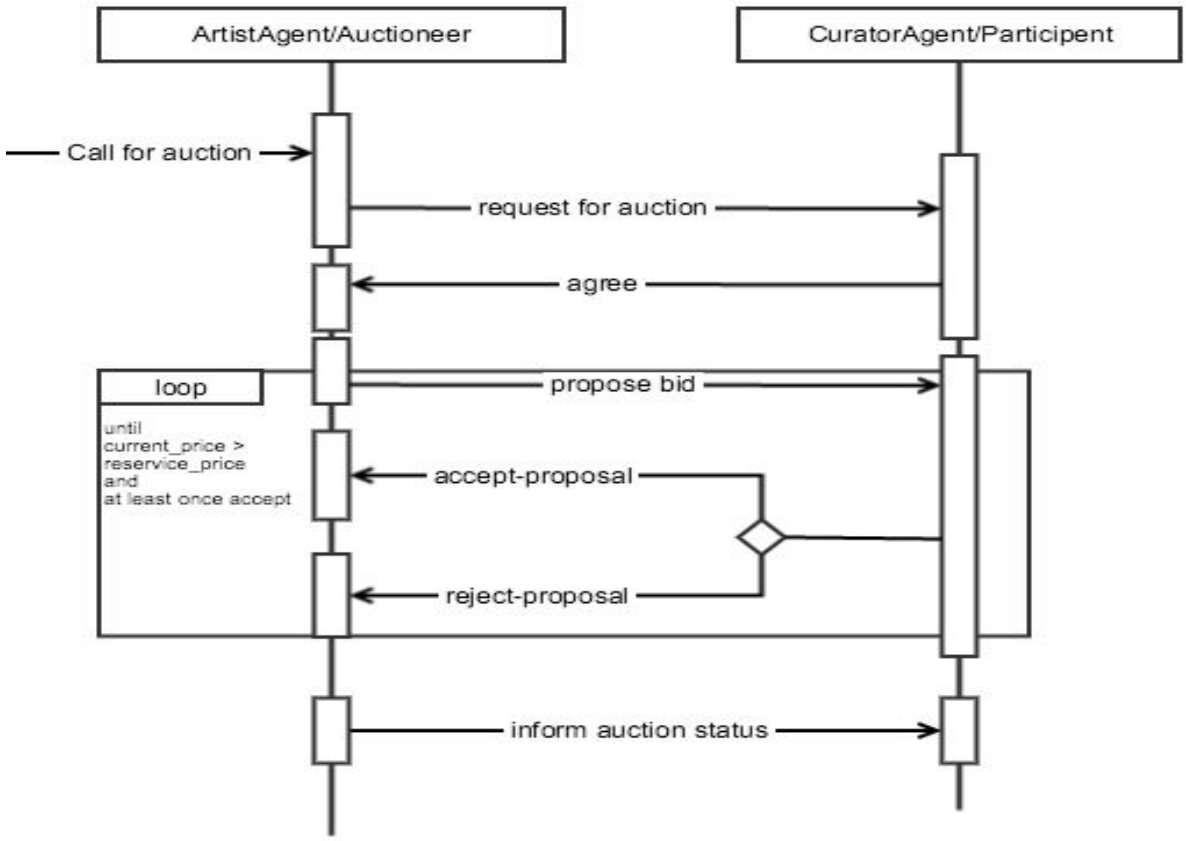
The participants who take the auction will reply with agree message. Then the Auctioneer propose a bidding price to all participant. The bidding price is reduced at each cycle until it gets an accepted bidding from a participant. The price is reduced according to an internal strategy of the auctioneer until it reaches the reserve price of the auction's item.

Interaction among agents

Sequence diagrams



The above sequence diagram shows the application of Recommendation Protocol in between Profiler and TravelGuide agents. When Profiler is recommended with tours, it contacts Curator agents to get more detail of tours' artifacts.

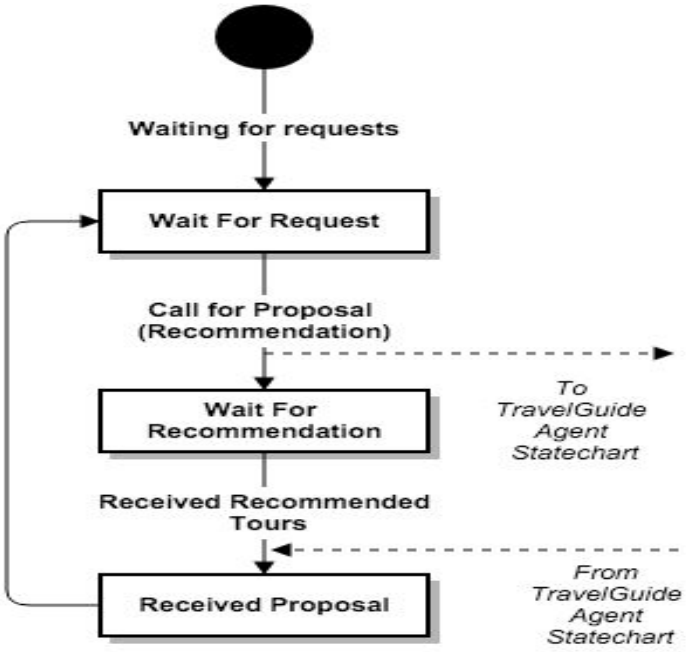


The above sequence diagram shows the application of Dutch Auction protocol in between Artist and Curator Agents.

Internal Agent Processing

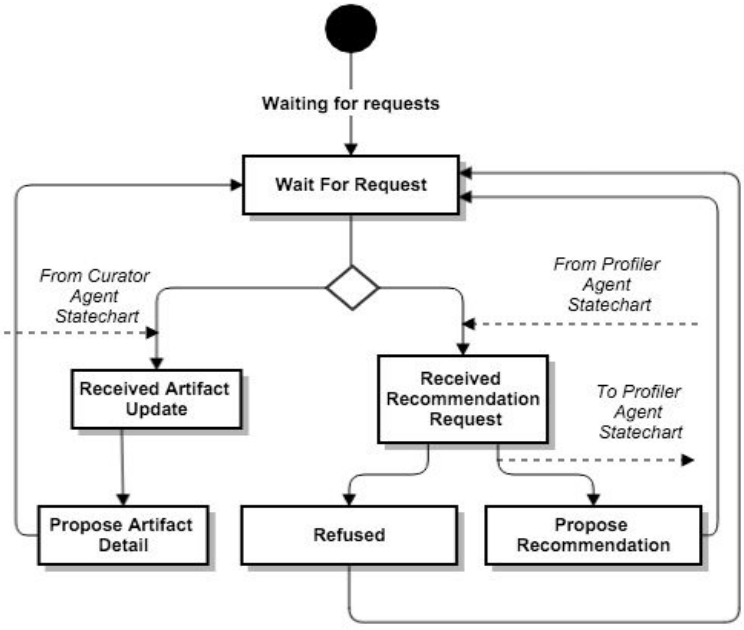
State chart diagram

Profiler Agent's Statechart



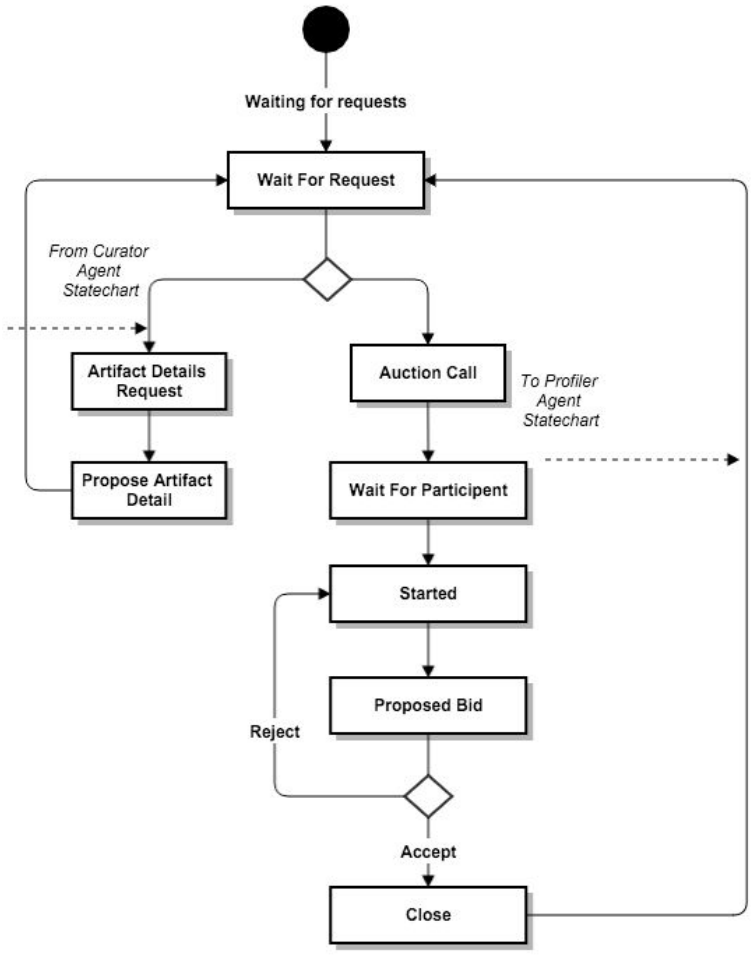
This is the internal processing of Profiler Agent. The Profiler Agent wait for requests from human user. On request it calls for TravelGuide Agent for recommended tours and wait for reply from Travel Agent. On receive of recommended tours it goes back to waiting for more requests.

TravelGuide Agent's Statechart



TravelGuide agent waits for request from Curator or Profiler agents. The request from Curator for update its internal knowledge of different artifacts. The request from Profiler is for recommended tours.

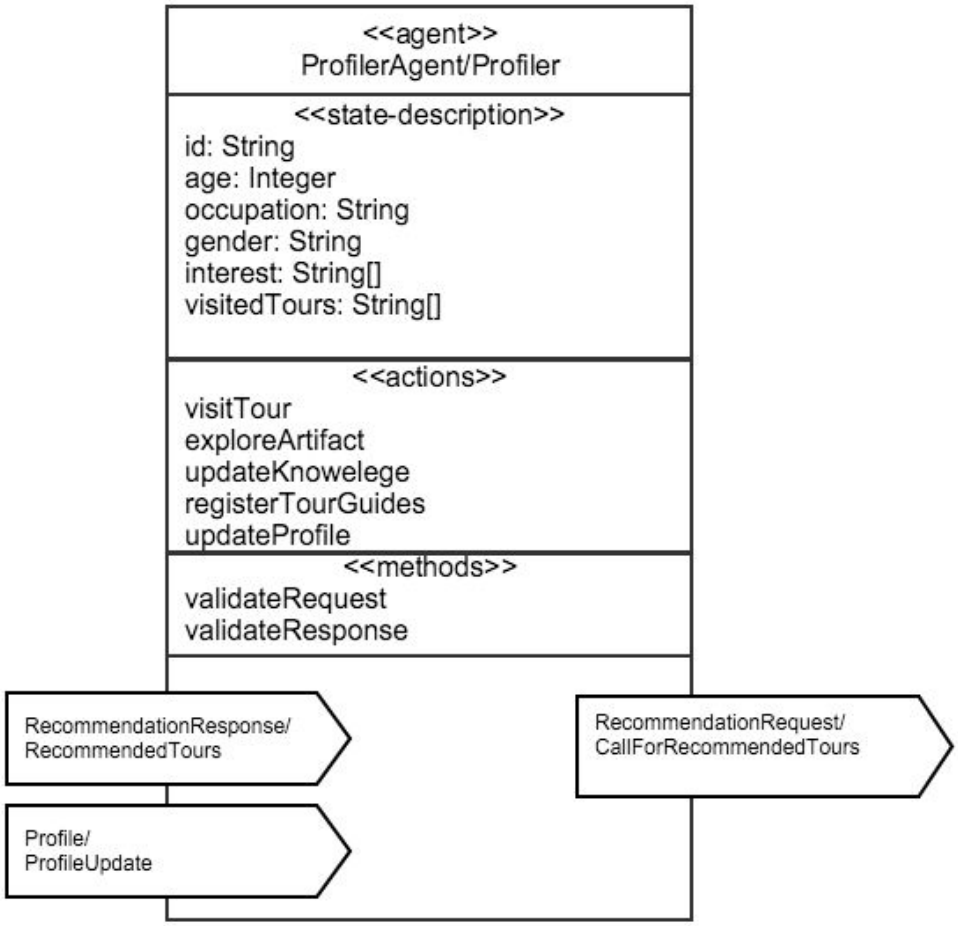
Curator Agent's Statechart

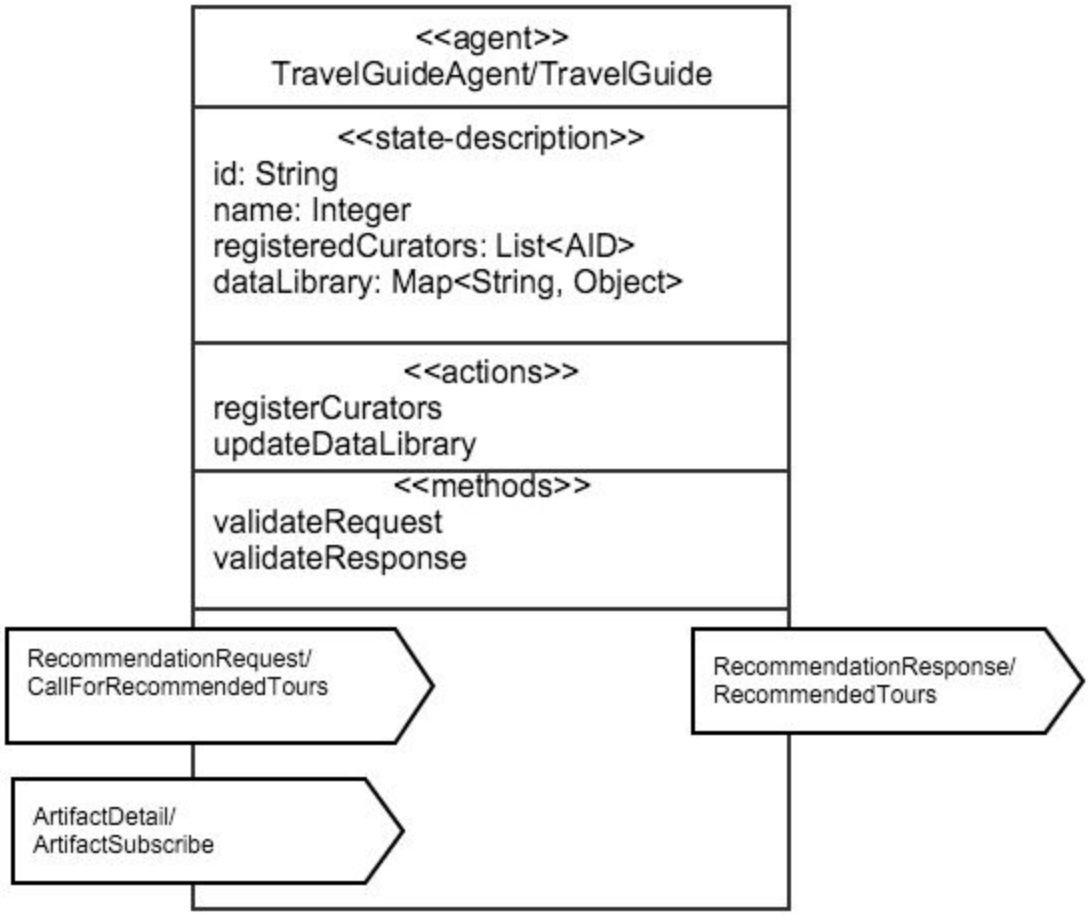


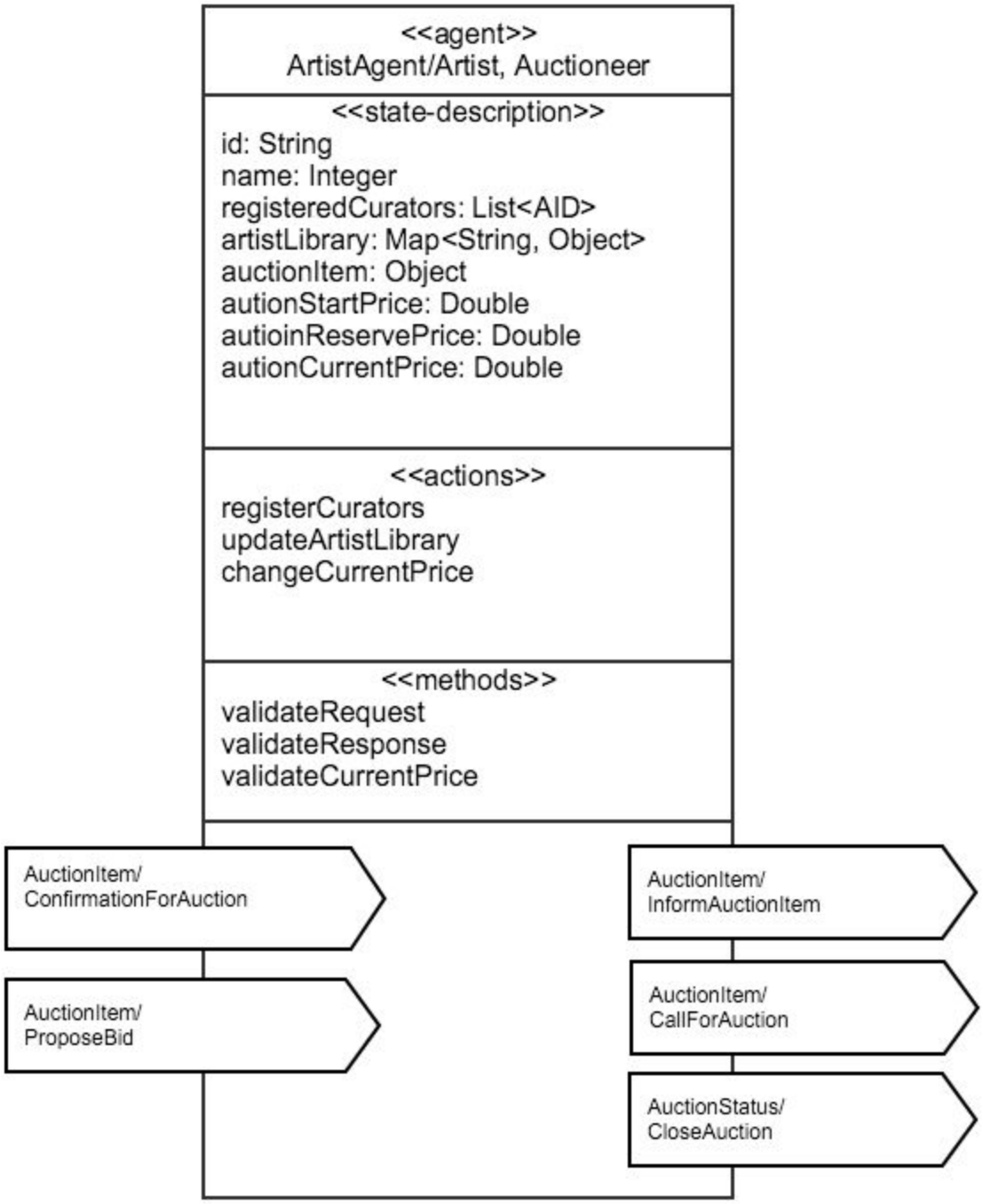
Curator Agent get request of Artifact details or mainly to start an Auction with profilers. On Auction, it call for an auction and wait for confirmation of participant from Profilers. The Auction is started and propose a bid. If the bid was rejected from all participant then new bid is offered based on internal bid price strategy. The auction will be closed on accept of a bid.

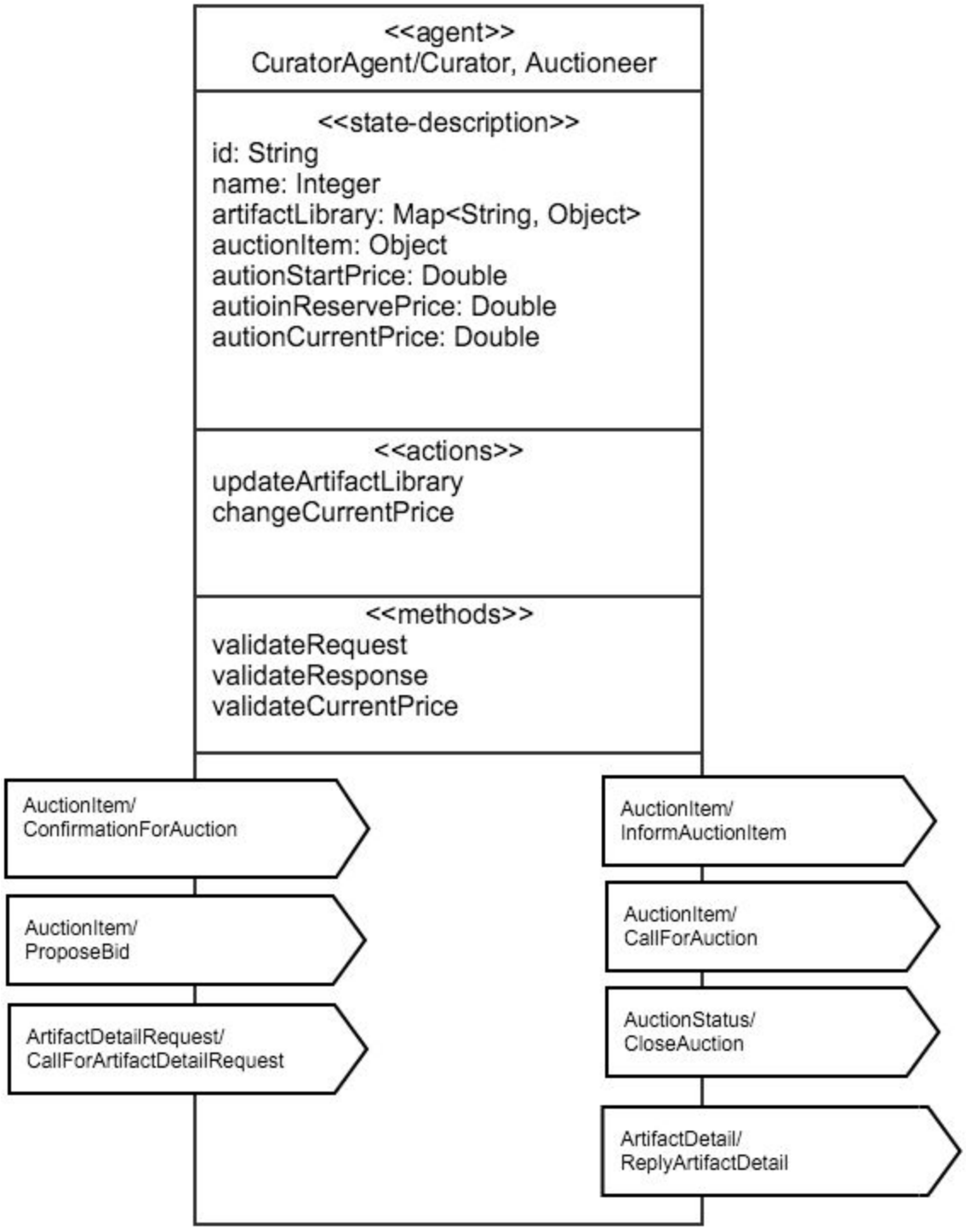
Task 3 - Behaviours in AgentUML

UML class diagram



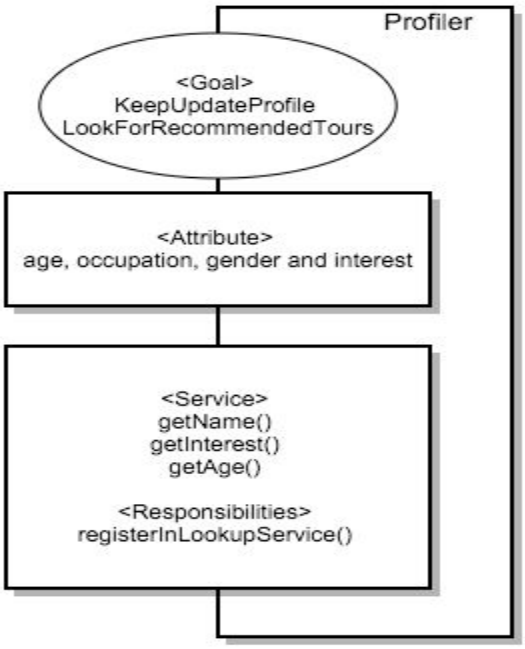

<<agent>>
TravelGuideAgent/TravelGuide
<<state-description>>
id: String
name: Integer
registeredCurators: List<AID>
dataLibrary: Map<String, Object>
<<actions>>
registerCurators
updateDataLibrary
<<methods>>
validateRequest
validateResponse
RecommendationRequest/
CallForRecommendedTours
RecommendationResponse/
RecommendedTours
ArtifactDetail/
ArtifactSubscribe

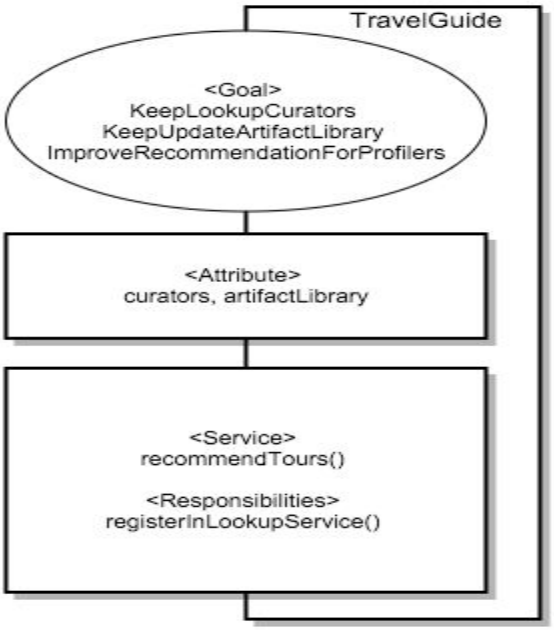

<<agent>>
ArtistAgent/Artist, Auctioneer
<<state-description>>
id: String
name: Integer
registeredCurators: List<AID>
artistLibrary: Map<String, Object>
auctionItem: Object
autionStartPrice: Double
autioinReservePrice: Double
autionCurrentPrice: Double
<<actions>>
registerCurators
updateArtistLibrary
changeCurrentPrice
<<methods>>
validateRequest
validateResponse
validateCurrentPrice
AuctionItem/
ConfirmationForAuction
AuctionItem/
ProposeBid
AuctionItem/
InformAuctionItem
AuctionItem/
CallForAuction
AuctionStatus/
CloseAuction

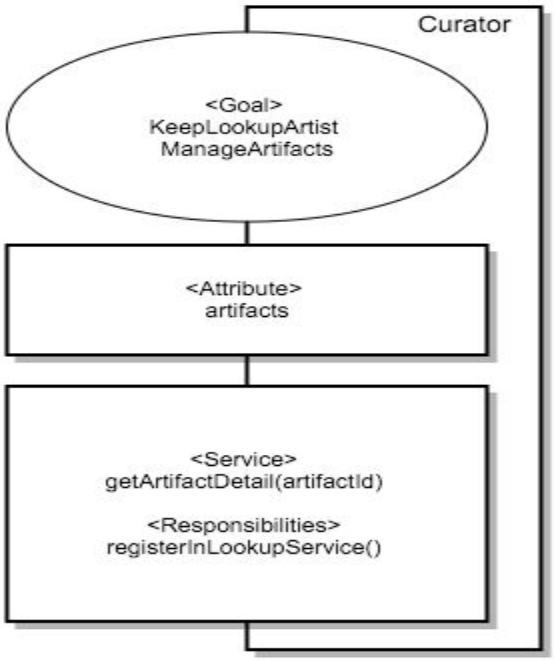

<<agent>>
CuratorAgent/Curator, Auctioneer
<<state-description>>
id: String
name: Integer
artifactLibrary: Map<String, Object>
auctionItem: Object
autionStartPrice: Double
autioinReservePrice: Double
autionCurrentPrice: Double
<<actions>>
updateArtifactLibrary
changeCurrentPrice
<<methods>>
validateRequest
validateResponse
validateCurrentPrice
AuctionItem/
ConfirmationForAuction
AuctionItem/
ProposeBid
ArtifactDetailRequest/
CallForArtifactDetailRequest
AuctionItem/
InformAuctionItem
AuctionItem/
CallForAuction
AuctionStatus/
CloseAuction
ArtifactDetail/
ReplyArtifactDetail

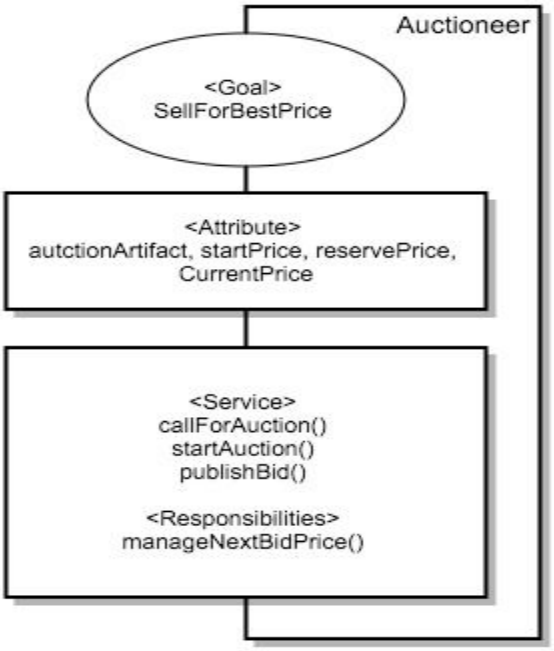
Task 4 - Role based modeling approach (ROMAS)

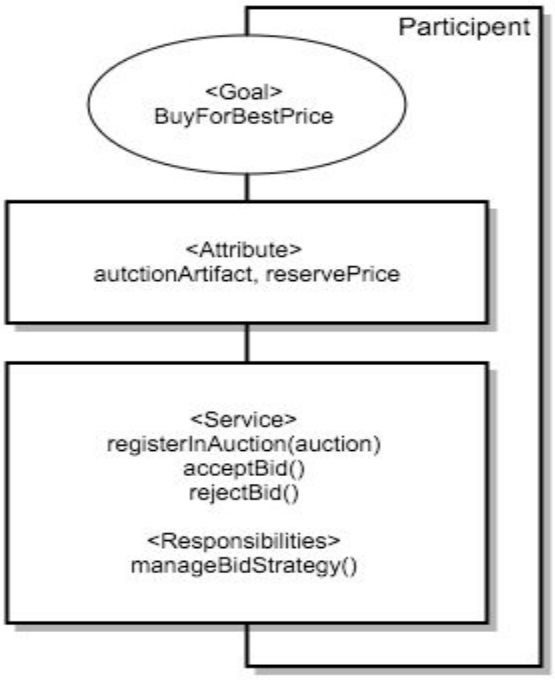
Roles Identification



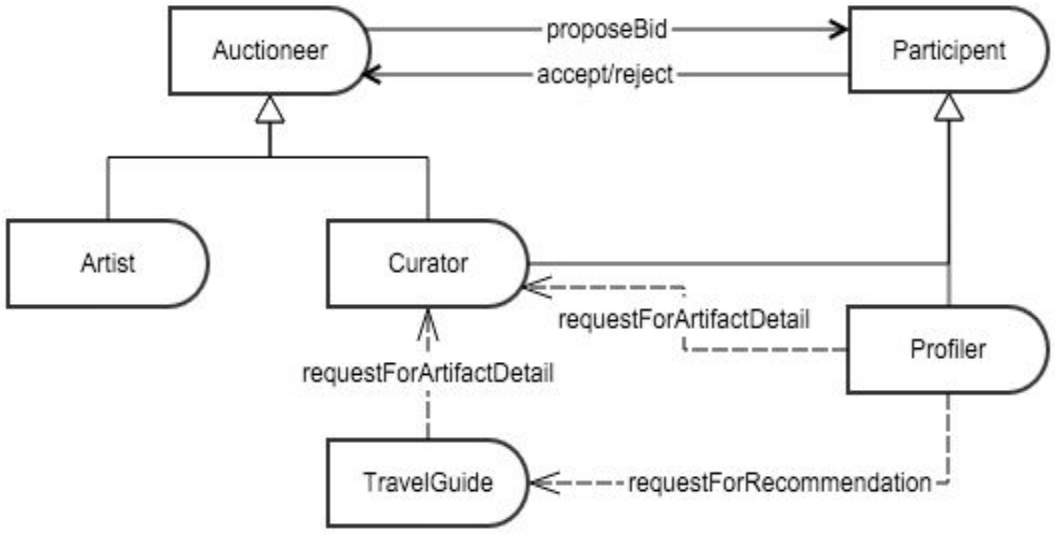




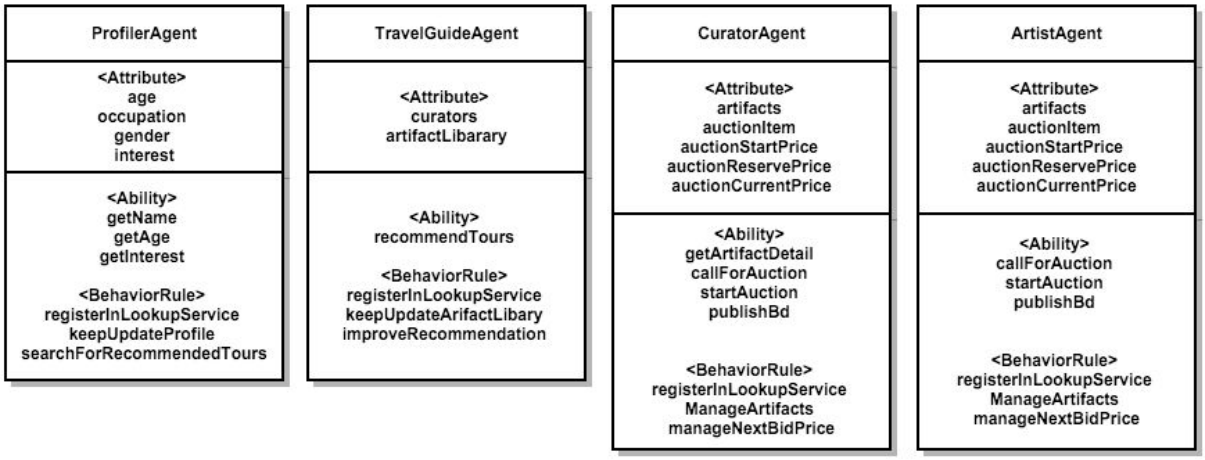




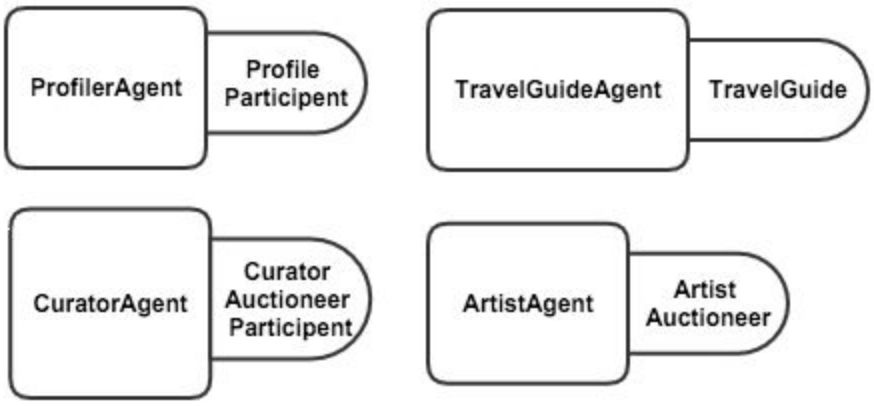
Roles Organization



Agents and Roles Mapping



Roles Bindings



GAIA Analysis and Role-based Modeling

- In RoMAS permission of state attributes is not considered as GAIA role analysis.
- Both model has Responsibilities of role as an attributes but Protocols is not considered in RoMAS. It is represented as a Service attribute.
- GAIA doesn't define the goal of agent, but in RoMAS.
- RoMAS has more simplified model of showing the relationship among roles, in a Role Organization diagram. But GAIS's role interaction model gives same means with more detail information.
- GAIA analysis process (defining Roles Model) is iterative. But RoMAS doesn't explain a such a processes.

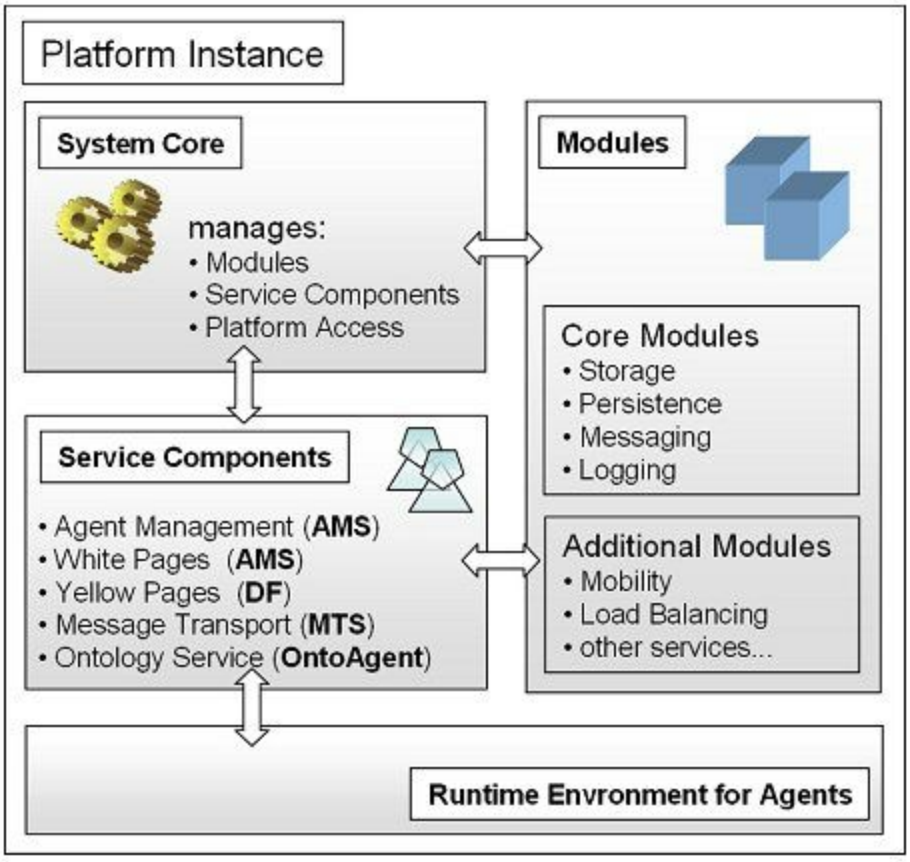
Task 5 - FIPA Compliant Implementation

AgentService
(http://www.agentservice.it/)

AgentService is an Agent Oriented Programming framework providing secure, efficient, extensible and modular environment to implement multi-agent system. It is portable over different CLI runtimes, Mono, Microsoft.NET and Shared Source CLI (SSCLI, aka Rotor). AgentService architecture is based two main core modules, Agent Platform and Agent Model.

Agent Platform

The platform provides with all the required services for multi-agents. It acts as virtual machine for the agents. The platform is designed according the reference implementation of FIPA, and with modularity design.



System Core

- Provides the basic services.
- Responsible for lifecycle of a platform instance, creating agent instances and resuming them from a persistent storage.
- And responsible for controlling the lifecycle.

Service Component

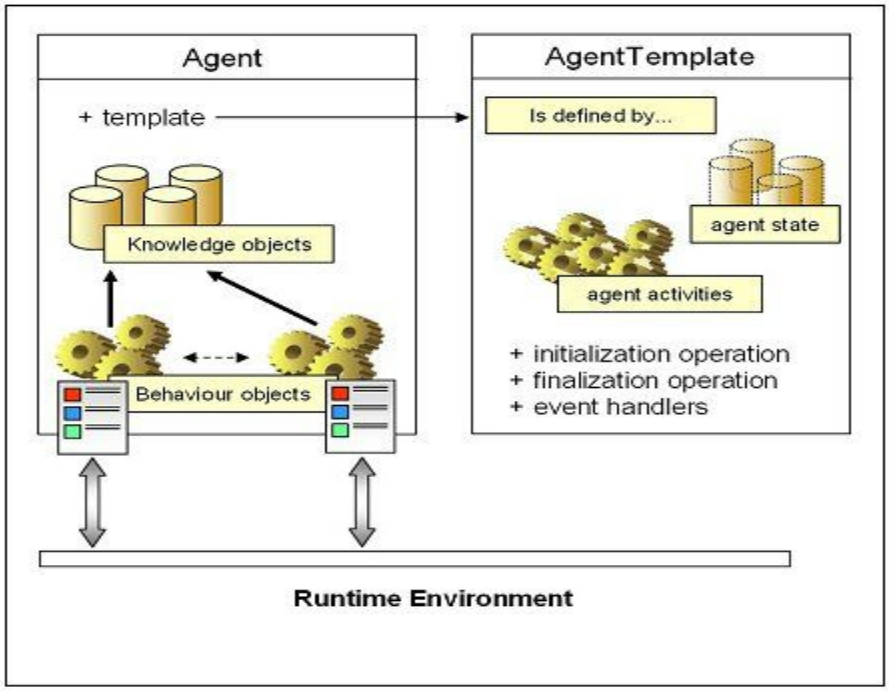
- FIPA compliant services.
    - Agent Management System (AMS)
    - Directory Facilitator (DF)
    - Message Transport System (MTS)

Modules

- Storage
- Messaging
- Logging

Agent Model

Agents in AgentService framework has two basic elements, behaviours and knowledge objects.



Agents interacts with Runtime Environment to access core services of the platform. E.g Message service.

Implementation Review

Define Agent

```
[AgentKnowledges(typeof(KnowledgeObject))]
[AgentBehaviours(typeof(BehaviourForSampleAgent))]
public class SampleAgent: AgentTemplate {
     public AgentOne() : base() {
     }

     public override void Initiate(IFactoryContext factoryContext) {
        factoryContext.CreateKnowledge(typeof(KnowledgeObject),
                                "KnowledgeForSampleAgent",
                                PersistenceMode.StrongPersistent,
                                'SampleValue');

      factoryContext.CreateBehaviour(typeof(BehaviourForSampleAgent),
                                "BehaviourForSampleAgent",
                                "KnowledgeForSampleAgent");

     }

     public override void Terminate() {

     }
}
```

Define Behaviour

```
public class BehaviourForSampleAgent: Behaviour {
     private KnowledgeObject knob;
     public BehaviourForAgentOne(string name,
                                KnowledgeObject knob): base(name) {
          this.knob = knob;
     }

     public override void Body() {
     }
}
```

JACK

(http://www.agent-software.com.au/products/jack/)

JACK represents a collection of autonomous agents that take input from the environment and communicate with other agents. Agents in JACK is defined with its goals, knowledge and social capabilities.

Agent Platform

Jack core platform is an extensible multi-agent runtime and manages the execution of message passing and reasoning. And it is provided with an agent specific language for defining Plans, the discrete reasoning executed by agents.
Jack serialize java object into human-readable ASCII text, something similar to YAML/XML.

Application

Norwegian Statoil
Norwegian-based Statoil uses JACK to support oil trading and operations management,

The Intelligent Prognostic Health Manager (iPHM)
AOS group develops an autonomous system that takes prognostics to the next level of capability.

The Improved Human Behaviour Representation (IHBR)
IHBR project predicts variation in decision-making time as well as errors that can occur due to moderating factors in Humans.

Implementation Review

JACK bundles with its own language for implementing Agents. It is a programming language used to describe an agent-oriented software system, which is a super-set of Java – encompassing the Java syntax while extending it with constructs to represent agent-oriented features.

Define Agent

```
agent AgentType extends Agent [implements Interface]  {
      #private data BeliefType belief_name(arg_list);
      #handles event EventType;
      #posts event EventType reference;
      #sends event EventType reference;

      #uses plan PlanType;
      #has capability CapabilityType reference;


}
```

Define Capabilities(Behaviour)

```
capability CapabilityType extends Capability [implements
InterfaceName] {

        #private data BeliefType belief_name (arg_list);
        #exports data BeliefType belief_name (arg_list);
        #imports data BeliefType belief_name ();

        #uses plan PlanType;

        #handles event EventType;
        #handles external [event] EventType;
        #posts event EventType reference;
        #posts external [event] EventType reference;
        #sends event EventType reference;

        #has capability CapabilityType reference;

}
```

Comparison with JADE

	JADE	**JACK**	**AgentService**
Semantic Web support	Low	Average	Low
Programming Language	Java	Java Jack Agent Language	Microsoft .Net, Mono, Shared Source CLI (SSCLI, aka Rotor)
Security	Strong Platform Security (User authentication, Jaas API) Signature and Encryption, HTTPS support	Strong Platform Security Authentication Domain protection mechanisms utilized in JDK	Moderate Platform security
Pragmatic overview	Command line installation High user support (FAQ, mailing list, defect list, API, docs)	Various installers available High User support(extensive documentation and supporting tools)	Installers available Moderate user support (FAQ, mailing list, defect list, API, docs)
Usability	User friendly Rich GUI	User friendly Useful GUI	User friendly Useful GUI
Specification	FIPA, CORBA	FIPA	FIPA
Communication	DCI network, TCP/IP	ACL (Asynchronous), MTPs,RMI, IIOP, HTTP, WAP	
License	Telecom Italia LGPLv2 Opensource	AOS Commercial and Academic license	l.i.d.o research group

References

http://jasss.soc.surrey.ac.uk/18/1/11.html
http://www.agentservice.it/