Distributed Artificial Intelligence and Intelligent Agents Project

KTH Royal Institute of Technology
School of Information and Communication Technology
Student:Fanti Machmount Al Samisti (fmas@kth.se)
Student:August Bonds (bonds@kth.se)

December 18, 2015

Contents

1 Introduction

Stub

2 Task 1 - GAIIA

2.1 Requirements Statement

General Commonality Requirements

- The curators and artifact manager should be fault tolerant
-

Self-Optimization Commonality Requirements

- The curator should only bid on profitable auctions.
- The curator should minimize the processing on every profiler query.
- The artifact manager should adapt the price setting and price reductions to the demand and item quality.
- The profiler should optimize bidding strategy according to client interests.
- The artifact manager should post multiple auctions if demand is high.
- The artifact manager should post more low-quality items for a high price if multiple items go unsold.

Self-Healing Commonality Requirements

- If there is a database corruption, artist manager should renegotiate with curators.

Self-Protection Commonality Requirements

- The artist manager should detect collusions and increase the minimum price.
- Every transaction should be atomic and persistent.

Miscellaneous Commonality Requirements

- bla

2.2 Roles model

Role Schema	Tourist
Description	Requests a tour based on the tourists' interests
Protocols and Activities	**protocols** FIPA.Request, **activities** parse and store the reply with the artifacts, initiate a tour as soon as the reply from the curators is parsed
Permissions	**reads** available curators from *DF*, **reads** tourists' interests, **changes** local artifact information
Responsibilities	
Liveness	$Tourist = (\underline{RequestTour}.StartTour)^{\omega}$
Safety	$interests > 0$

Role Schema	Gallery monitoring
Description	Monitor the artifacts database, perform maintenance tasks on it and reply to tour requests
Protocols and Activities	**protocols** Fipa.INFORM, **activities** perform maintenance tasks on the artifact DB on a regular basis, monitor for problems on artifact DB
Permissions	**changes** artifact database, **reads** artifact database
Responsibilities	
Liveness	$Monitor = \underline{Reply} \star \lvert\lvert \underline{MaintainDB}^{\omega}$
Safety	$databaseIntegrity \& noOfRequests < 30$

Role Schema	Bidder
Description	Participate in an ongoing auction
Protocols and Activities	**protocols** Dutch Auction, **activities** decide on whether to bid according to his strategies
Permissions	**reads** artifact price
Responsibilities	
Liveness	$Bidder = \underline{BidInAuction}^{\omega}$
Safety	$interestedInAuctionedItem$

Role Schema	Artifact adder/updater
Description	Add in the database or update an item with a newly bought artifact from the auction
Protocols and Activities	**activities** add new items to the artifact DB
Permissions	**changes** artifact database, **reads** artifact database
Responsibilities	
Liveness	$Adder = (WaitForItem.\underline{AddInDB})^{\omega}$
Safety	$\forall itemInDatabase = unique$

Role Schema	Auction Initiator
Description	Start and coordinate a new auction
Protocols and Activities	**protocol** Dutch Auction, **activities** decide the price of the next item to be auctioned based on its earnings and strategies
Permissions	**reads** artifacts for auction, **changes** available artifacts for auction
Responsibilities	
Liveness	$Initiator = (\underline{ChooseItemAndPrice}.GetBidders.Inform)^{\omega}$
Safety	$auctionProfit > 0$

2.3 Interaction model

2.3.1 Tour

Figure 1: Tour interaction between a Tourist and one of the GalleryMonitors

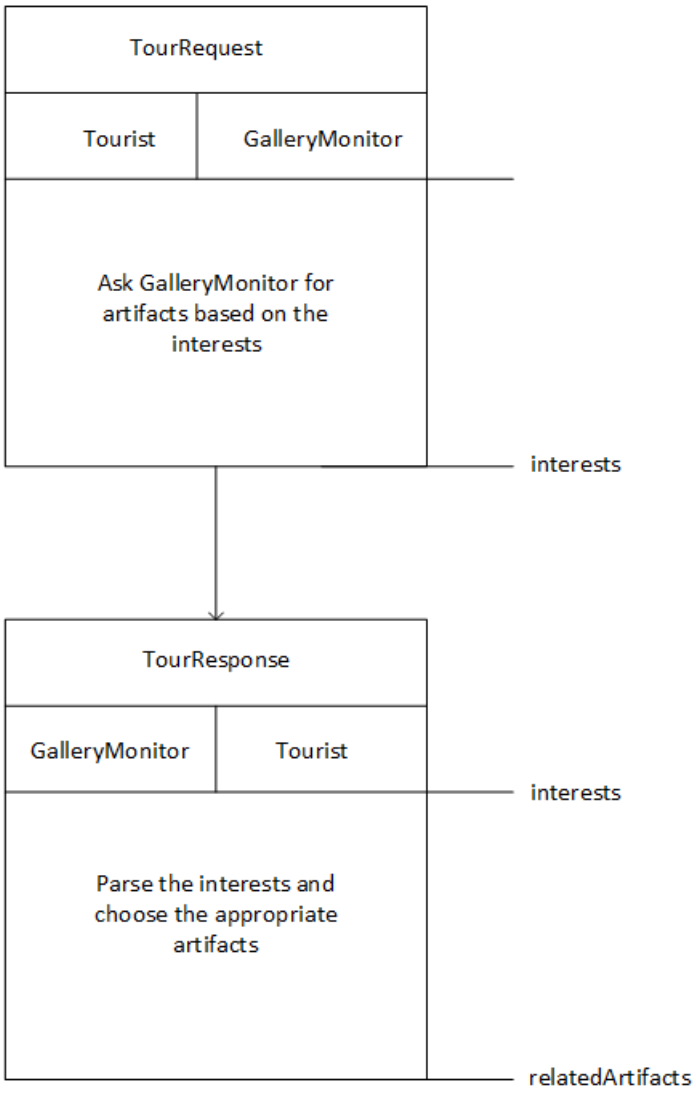


Figure 2: Auction initiation: If the bidder wont participate, this interaction ends here.

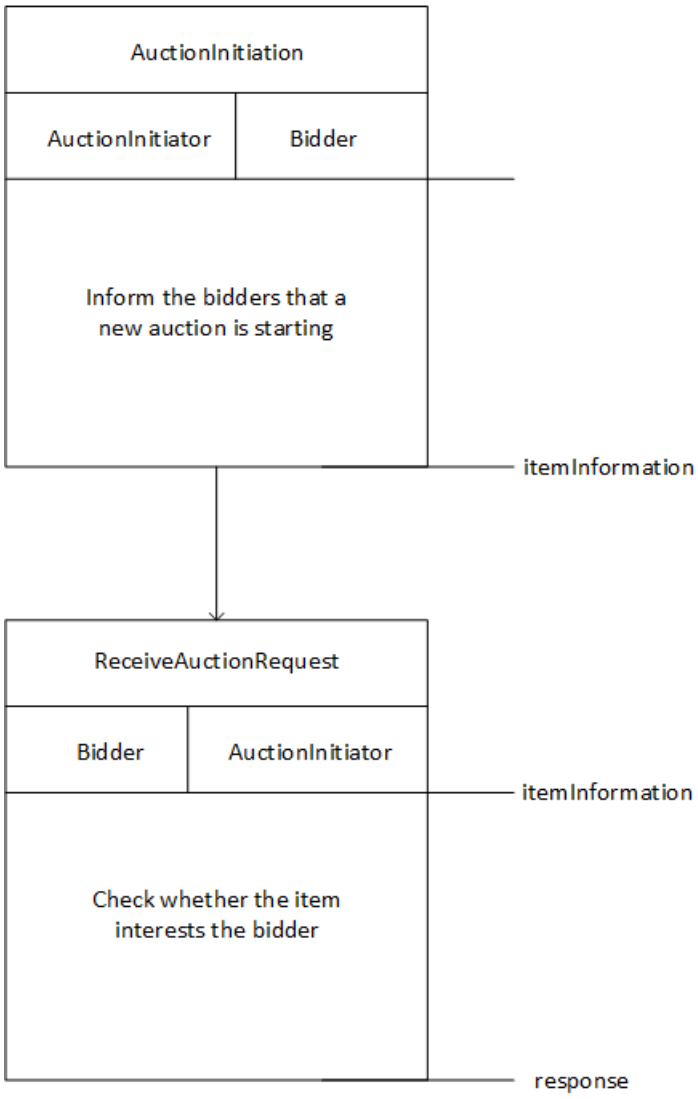
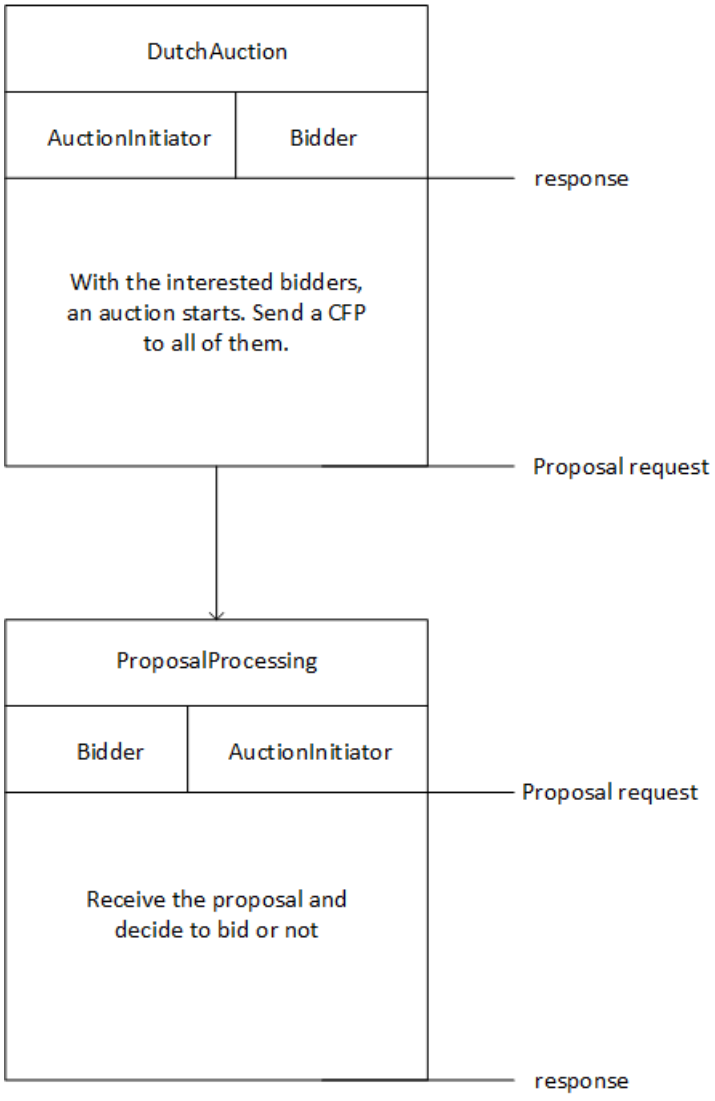
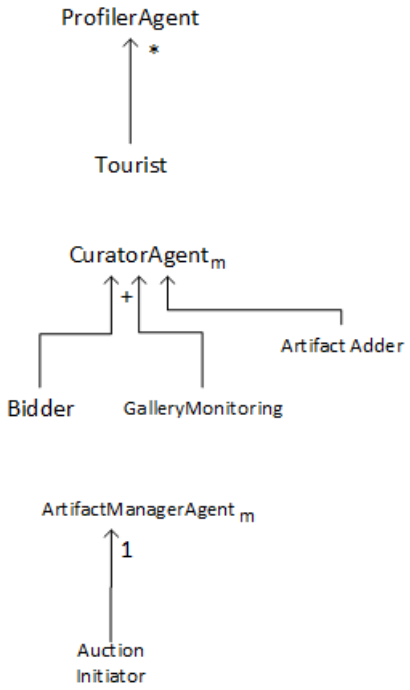


Figure 3: Start an auction and the bidders decide if to bid or not



2.4 Agent model

Figure 4: Actors and their relation to the roles



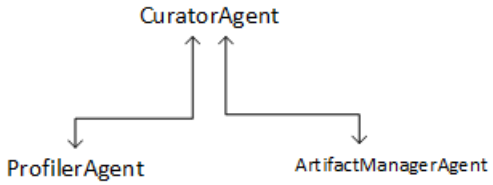
2.5 Service model

Table 1

Service	Inputs	Outputs	Pre-condition	Post-condition
proposeTour	interests	artifact list	interests >0	profiler has a tour
startAuction	available-Artifacts	itemToAuction	available-Artifacts >0	true
complete-Auction	bidders, item	outcome	bidders >0	winner $\vee$ low price
bid	item	decision	true	bid $\vee$ reject proposal

2.6 Acquaintance model

Figure 5: Actors and their communication



2.7 Mobility model

Table 2: All places that tourists can visit and auctions can take place

Place Types	Description	Instances
museo galileo museum	A place to see items that comprise a major collection of scientific instruments. Also auctions for relevant artifacts happen to enrich the collection of the museum.	1
Heritage Malta museum	A place to see items that are protected by Maltese national agency for museums. Also auctions for relevant artifacts happen to enrich the collection of the museum.	1

Table 3: Agents and their relationship to the places

Agent Types	Mobile	Place Types	Constraints
PrrofilerAgent	-	-	-
CuratorAgent	√	All	1 in each place
ArtifactManager Agent	√	All	1 in each place

Figure 6: Actor-Place cardinality

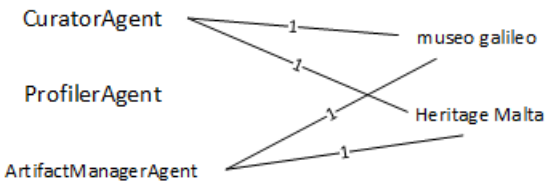


Table 4: Curator path description

Agent	CuratorAgent
Description	An interested curator will move to a museum place to take part in the auction that its been held there.
Origin	Any musuem in the world
Destination	museo galileo or heritage malta
Atomic movements	Move from the original museum to the destination
Paths	(Original Museum - Destination - Original Museum)*

Table 5: ArtifactManager path description

Agent	ArtifactManagerAgent
Description	An artifact manager will move to a museum place to organize the auction that its been held there.
Origin	Any musuem in the world
Destination	museo galileo or heritage malta
Atomic movements	Move from the original museum to the destination
Paths	(Original Museum - Destination - Original Museum)*

3 Task 2 - AgentUML

4 Task 3 - UML and Behaviours

5 Task 4 - ROMAS

6 Task 5 - Framework comparison

https://en.wikipedia.org/wiki/FIPA
Options(Choose 2)
Looks too corporate-y - http://www.agent-software.com.au/products/jack/
JIAC - IJava framework - http://repositories.dai-labor.de/sites/jiactng/5.1.5/
SPADE - Python framework - https://github.com/javipalanca/spade
JADEX - https://www.activecomponents.org/bin/view/About/Features

My choice would be the SPADE and JADEX.

We have to compare:

- Architecture of platform
- Services provided by it

- Comparison of implementaBon of a simple scenario same as Question 2 (i.e. Service Implementation, Service Registration and Service Discovery)
- Notable projects using it
- Personal opinion on practical issues compared to JADE, we could use point 3 as a starting point.

FIPA compliance:

- Agent Communication Channel(ACC): A mechanism which allows the platform and the agents in it to communicate with each other.
- Agent Management System(AMS): A way for the agents to be registered in the platform and be reachable for contact.
- Directory Facilitator(DF): Agents have the ability to publish services they offer, essentially yellow pages.
- Agent Communication Language(ACL): Common agent language, two possible syntaxes XML or Lisp based.

6.1 SPADE

Agent platform based on XMPP and Jabber. User(agent) and Server(platform). Agents can be in any programming language if they conform to XMPP protocol.

XMPP: XML inspired protocol for instant messaging and presence information. Jabber protocol at its core follows XMPP. It is an open standard and extensible. Noteable uses: Pidgin, Google Talk, iChat.
Features:

- Open, public, free
- Asynchronous: If user online guarantee of delivery, if offline store and forward message until the client reconnects. Ease of talking with non-human systems.
- Decentralized: Anyone can have a server, in fact its a server network, choice of whether to trust a server.
- Secure
- Extensible

- Flexible: XML messaging based tech has a myriad of uses, IM is just one of them. Jabber applications beyond IM include network management, content syndication, collaboration tools, file sharing, gaming, remote systems monitoring and, now, agent communication.

SPADE Agent Library
Module to build SPADE agents that work with the SPADE Agent Platform.

SPADE Agent Model
Composed of a connection mechanism to the platform, a message dispatcher and a set of agent behaviors. Each agent is recognized by a JID(Jabber ID) that looks like (name@host, password) and by an address(xmpp://acc.myprovider.com). Each agent registers to the server as a jabber user(mandatory, not like DF in JADE) which opens a longlived communication stream.
A message dispatcher is responsible for receiving, delivering to a behavior queue(MessageTemplate) and sending messages.
Simultaneous behaviors: Cyclic, OneShot, Periodic, TimeOut, FSM and Event.
Message sending: like JADE
A SPADE agent cannot send nor receive messages until its behaviours are active. That is, do NOT place calls to the send or _receive methods inside the _setup and takeDown methods.
Search agent service, Modify Service(update your info in the AMS)
DF: same as Jade
FSM: same as Jade
Event Beh. : The main difference between an Event Behaviour and, say, a One-Shot Behaviour is that the Event Behaviour is not instanced nor is it running until the trigger event happens.
The SPADE BDI Agent Model: Belief-Desire-Intention:

- Belief: knowledge
- Desire: goals
- Intention: the way the agent has decided to achieve his goals, Plans.

SPADE deviates from this by using Service Oriented Computing(SOC) together with dynamic compilation of services in SPADE, which we have called Distributed Goal Oriented Computing.
SPADEs BDI model:

- Belief: Agent knowledge base, insert, delete, make queries.
- Goals and Desires: When an agent expresses a Goal, it means that the agents wishes to accomplish the expression contained in such Goal. When a goal is selected for accomplishment, it becomes active.

- Services: Method offered by the agent to the rest. Services can be composed into a sequence forming the Plans. Services have in their description both a pre-condition (P) and a post-condition (Q). The pre-condition P represents a state of knowledge that must be present in order to execute the Service. The post-condition Q represents the state of knowledge that the agent will achieve once the Service has been invoked.(Like MMSE ocl)
- Plans: Sequence of services to achieve the goals. Agents use plans to achieve their goals. Also they have their own Pre and Post conditions. Services composing a Plan's actions do not necessarily belong to the same agent.

```
Name = "agent@myhost.myprovider.com" . Addresses = ["xmpp://agent@myhost.myprovider
```

Knowledge base: Prolog based

```
$ agent.saveFact("MyList",[5,6,7,8])
$ agent.getFact("MyList")
> [5,6,7,8]
```

Plans and Services: An agent exposes a method unlike JADE

```
#This is the method executed when the service is invoked
def s1_method(Value):
return {"Myoutput1":1} #the return value is a dict containing Facts in the form name

#Create the service profile
s = DF.Service(name="s1", owner=agent.getAID(), inputs=["Value"],outputs=["O1"],P=["

#Finally register the service
agent.registerService(s, s1_method)

        agent.addPlan(inputs=["Value"],outputs=["O2"],P=["Var(Value,0,Int)"],Q=["Va
```

GOALS:

```
g = Goal("Var(O1,1,Int)")
$ def goalCompletedCB( goal ):
print "Goal completed!"

$ agent.saveFact("Value",0)
```

```
$ agent.setGoalCompletedCB( goalCompletedCB )

$ agent.addGoal( Goal("Var(Value,0,Int)") )
> "Goal completed!"
```

BDI: During the agent execution, classic SPADE behaviours can coexist with the BDI model. Every time that a new Goal is introduced into the agent, it will try to achieve it looking for a Plan that fits the task. All this work happens in a completely transparent way for the user.
Easier to work with knowledge base than JADE and its done in a declarative way.

Agent platform
Like JADE we start a server that acts as the main point/base of the agents.

6.2 JADEX

https://www.activecomponents.org/bin/view/Documentation/Overview
https://www.activecomponents.org/bin/view/About/Features

7 Evaluation

Stub