Educational System for Agent-based Distributed Task Processing

Subhabrata Ghosal, Dusan Sormaz, Arkopaul Sarkar
Department of Industrial and Systems Engineering
Ohio University
Athens, Ohio 45701
Email: sg369412@ohio.edu, sormaz@ohio.edu, sarkara1@ohio.edu

Abstract
The growing research on Agent based computing has opened a new vista towards better functioning of distributed systems. The interoperability and autonomy of intelligent agents provide a convenient and powerful solution for establishing communication within a distributed network. Agents are nothing but software abstractions which have innate ability to work independently, communicate with other agents with the help of a specified ontology and relocate their execution into different computing systems. In this paper, we propose to set up an educational system using a multi agent framework, where intelligent mobile agents are used for different task processing in a distributed network. These agents which require different services for the completion of a job are called Task Agents. These task agents request service remotely from a Service Agent. We have demonstrated this framework to implement mathematical operations on geometric shapes in a sequence of operations carried out by the agents. The sample run is executed from multiple remote locations carrying out varying sequence of operations. This prototype can be used for various engineering applications involving distributed model like Communication Networks, Industrial Control Systems, Parallel Computing, Manufacturing etc.

Keywords: Multi Agent Framework, Mobile Agent, Distributed Processing

1 Introduction
A distributed system is a collection of devices operating from remote locations autonomously and connected by a network having a distributed middleware which helps them to keep record of events and execute a task as a whole entity. There are various parameters which control the efficiency of a distributed system like network traffic, latency etc. Also the behaviors of the entities of a distributed system majorly affect the performance of the system. This brings us to the concept of a framework of mobile multi-agents which can very effectively create a distributed network with reduced network traffic, latency etc. The autonomous behavior of the agents and their independent existence is useful to construct a stronger and fault tolerant distributed environment. Distributed Processing in general happens through various communication protocols in which multiple communications are involved which increases the data volume in the network and can jam it. However for a multi agent framework, the entire agent can relocate in the environment which saves multiple communications and decreases chances of jam. Further, as the mobile agents are able to relocate, the processes which earlier used to happen through network can now be achieved in a remote system, which saves a lot of Network bandwidth. This paper is organized in the following way. In section 2, we have studied the earlier works on using a multi agent framework for distributed processing. Section 3 provides the methodology of development of the multi-agent education system, Section 4 deals with the implementation in a network of multiple remote systems. Finally in Section 5, we discuss the results and the scope of future work on this area.

2 Development of Distributed System using Multi-Agent Framework – Previous Works

The evolution of Agent Based systems [1] can be traced back from the origin of MAS or Multi Agent system which is a product of Distributed Artificial Intelligence. The need to implement mobile agents in distributed computing can be observed from the very beginning which encouraged more research work on this domain and various intelligent agent frameworks like DAML, Jason, OWL, JAT etc. have been tested since then. The utility of a multi-agent framework in distributed or parallel processing was explained very lucidly by Danny Lange in [2]. He explained that the major reasons behind using mobile agents for distributed networks is the amount of autonomy and the asynchronous activity of the agents along with the low network usage required for this framework. One of the earlier researches in the field of distributed processing to automate the design and manufacturing process involves utilizing a multi-agent system to build a holonic manufacturing system [3], which is an intelligent shop floor management technology. In this model, intelligent mobile agents called "holons" interact among each other to configure the shop floor and take intelligent manufacturing decisions. Shen and Norrie, in their paper, "Agent-Based System for Intelligent Manufacturing" [4] discussed vital issues towards implementation of multi-agent technology for developing manufacturing systems like supply chain management, distributed dynamic scheduling etc. Application of Agent-Based system in Process Planning has been demonstrated by [5], where a dynamic process planning system has been developed to facilitate dynamic changes and automatic update of process plans. Sormaz in his previous works have concentrated on distributed modeling and interactive process planning using IMPlanner [6] for efficient configuring and customization of products. In this paper he has designed integrative model for the product design using distributed data processing between different hierarchical object models to find out the most efficient one for product designing. In this paper he also focused on service oriented agent organization, the concept of which will be later proliferated in this paper. In another paper [7], he proposed a flexible agent based framework to address the process sequencing issue. This framework was built using a tool called space searcher which uses a search algorithm to obtain context specific optimal process sequence. Further he worked on the development of a distributed multi agent framework for design and manufacturing integration [8]. Here Sormaz et al. has introduced concept of autonomous agents in the form of task agents and service agents performing design and manufacturing tasks.

3 Agent Based Distributed Processing Architecture

When we are trying to set up an effective distributed network, we look into creating a system which has resource sharing, openness, concurrency, fault tolerance and transparency. The choice of Agents for distributed systems is guided by some very specific reasons. The autonomy of agents and their asynchronous way of communication has proven to be more robust than the standard master slave relationship model. If a task requires continuous open connection then it is very difficult for a fragile network system to carry on the objective. However for agent based system, the tasks can be embedded into mobile agents which can be transferred in the network thus being independent of the origin and can work asynchronously and independently. In that case the agents have the right to accept or reject any task forming an executive dependency relationship where a task can be rejected based on some definite unavoidable factor. Agents can also follow obligation dependency relationship where they will not be able to refuse any task but they can delay the tasks or schedule the tasks according to its own rule. Agents need not always be mobile. The agents which are providing

service and not seeking service from other agents can communicate with the surroundings through conventional means.
In this educational tool for agent based processing, we have shown both types of agents. The agents providing various services in a scattered distributed network are autonomous static agents. They are called Service Agents. The services are requested from these service agents by Task Agents which are remotely located and they automatically find the service agents as required. The service agents are in obligation dependency relationship with the Task Agents where they receive the various service requests from the task agents and synchronize them according to their own availability. The task agents sequence the various requests in order before starting with the first request and once one service is completed the finished result is automatically transmitted to the next service.

Figure shows the basic structure of agent based programming in which the Task Agents and the Service Agents exchange data in which the task agents sends data to the service agents which uses it to execute the service and return it to the task agents.

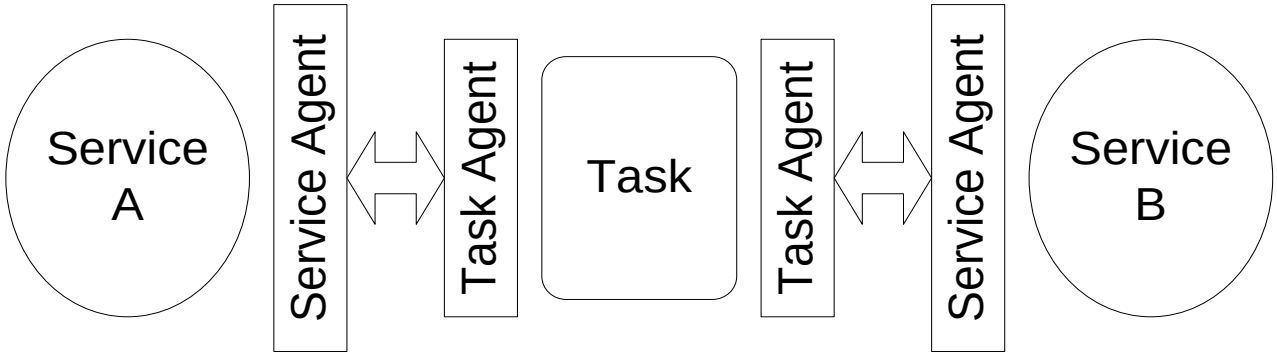

Figure 1- Basic Structure of Agent Based Programming

The various characteristic features of Task agent and Service Agent are discussed as follows. The main action of the Task Agent is to search for the server which will accept the job in sequence .The transferring of data happens through a series of services located remotely by searching their Internet Protocol address. The Service agent's main function is to process the jobs which are waiting in their queue. The Service agent follows the normal first in first out queue policy. The Task Agent on the other hand follows the policy of a rule based system

The task agent contains contents along with the IP address and port number of the various service agents it is going to request service from, arranged in sequence of services. The service agents on the other hand also contain contents which will interact with the content coming from task agents for the execution of services. The service agents maintain a history of the services already performed and the present tasks which are in queue. The task agent's lifecycle starts with the "*uninitiated*" state after which it is born, then starts searching for various services and once it finds the Service agent it goes to the "*in-transit*" state, then after entering the queue it goes in the waiting state. Finally it goes in the "*in-service*" state when the Service agent starts working. These cycle continues till all the services are completed after which the task Agent obtains the "*complete*" state. The Service agents on the other hand just maintain two states, "*idle*" after finishing a service and "*busy*" when it is performing a service.

The behavior and state changes of Service agents and Task agents are explained through the following sequence diagrams in Figure 2

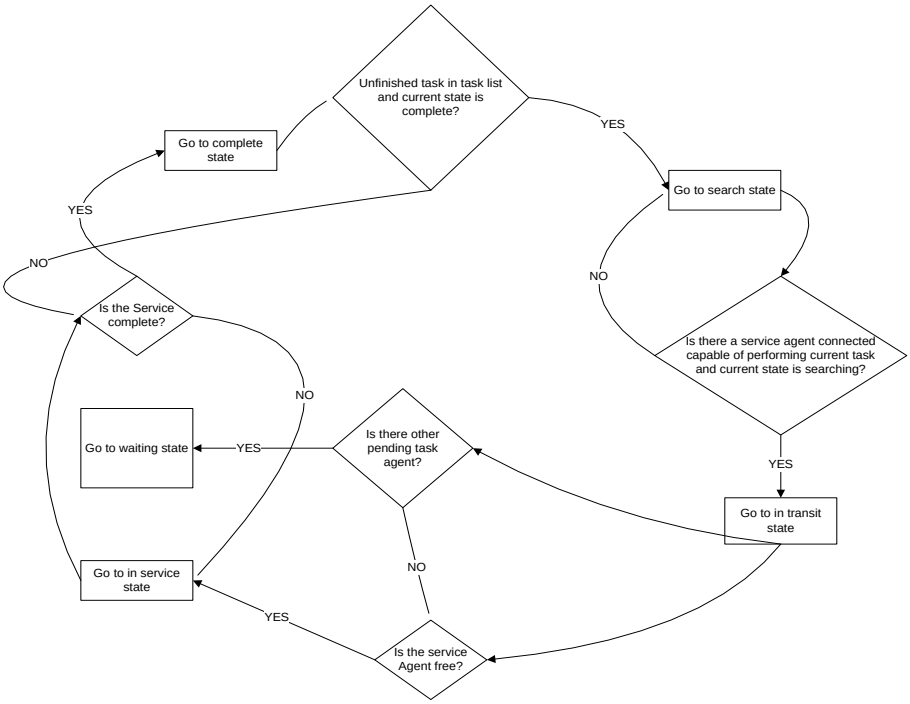

Figure 2A- Task Agents Sequence Diagram

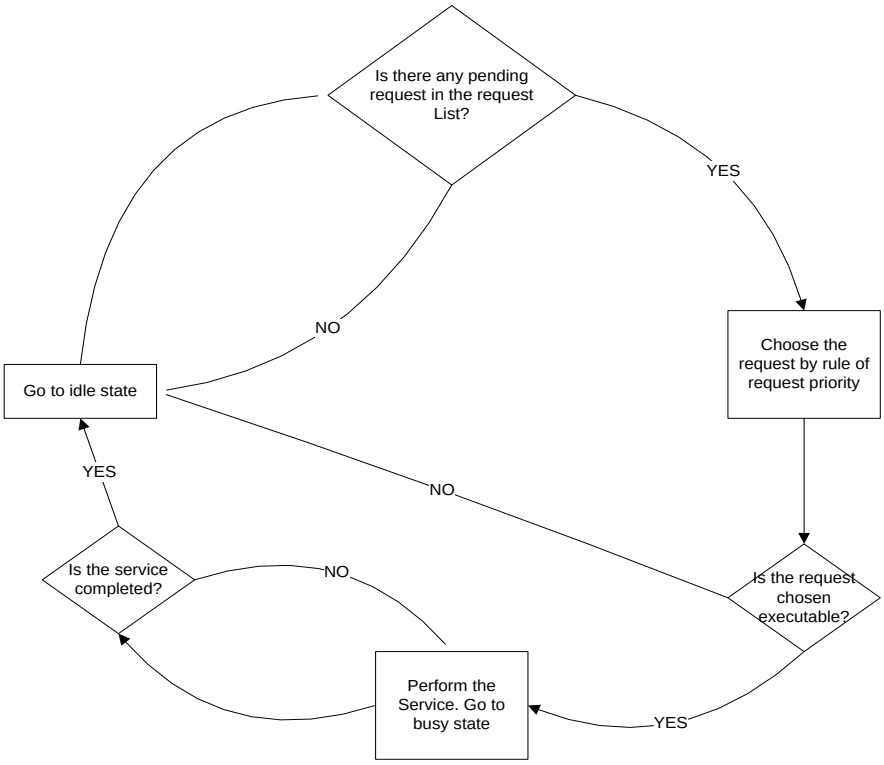

Figure 2B- Service Agents Sequence Diagram

4 Educational Tool for Distributed Multi-Agent Framework- Creative Geometry Implementation

The objective behind creating the educational tool is to make a lucid understanding of distributed processing using multi-agent framework. The goal of this system is to demonstrate explicitly the working of intelligent agents in a distributed network, their advantages and limitations. We have selected the domain to modify various geometric shapes using intelligent agents. This domain is selected because it is simple and easy for understanding the whole distributed processing and also it provides visual representation of the processing and results. A task agent called GraphAgent contains a collection of different tasks on geometric shapes which have to be performed on an initial shape. These tasks are sent in sequence to various service agents, each of which can again create different geometrical shapes which interact with the shapes coming from the task agents to perform various operations. The operations which are performed by this educational tool are union, intersection and difference. The overall structure of the tool is shown in Figure 3.

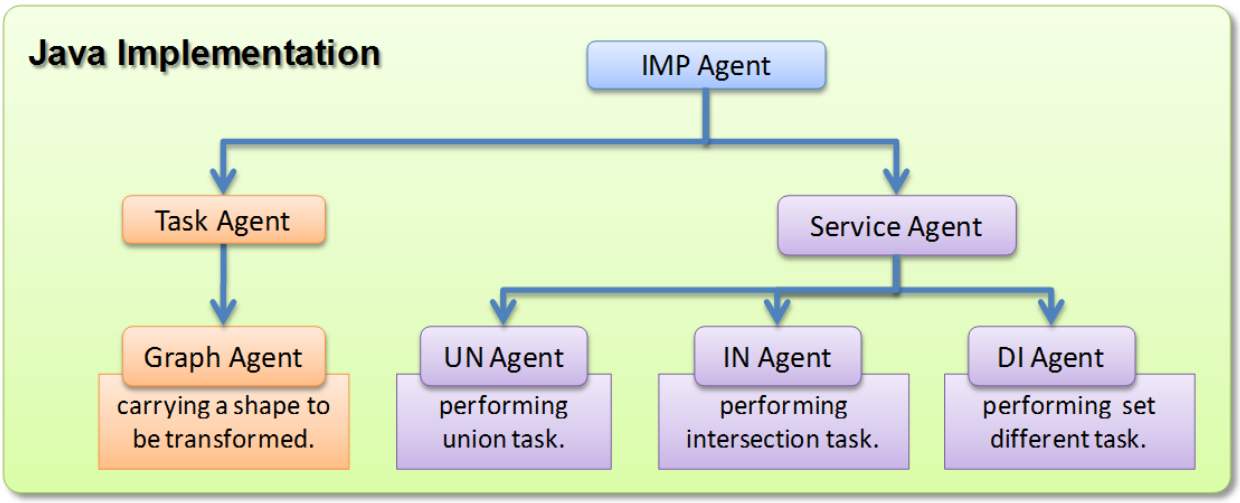

Figure 3- Overall Structure of Educational Tool

4.1 Architecture

The tool is implemented in Java where it is constructed following the service based architecture which is already discussed. The two main root classes are IMPAgent and IMPTask.

- **IMPAgent:** IMPAgent is the abstract class from which all agent classes are extended. The core properties of the agents are integrated in this class which includes the name, host address, and the state. The state of each agent is again derived from the class called the IMPState which maintains all the states of the agents which are discussed earlier. The abstract method for creating the basic user interface is also defined in this class. The IMPAgent is extended by TaskAgent and the ServiceAgent class. The TaskAgent is the parent class of the GraphAgent which works as the task agent for this tool. TaskAgent contains the common properties of any task agent which includes creating the task, and maintaining the list of task sequences, searching for the service agents and connecting with them, creating the graphic user interface for the task agents. The GUI is created using Swing components in Java. The geometric shapes are created inside the GraphAgent class by

creating a specific panel in the GUI for task agents and the shapes are derived from a java project created for generating various geometric shapes called the DrawProject. The ServiceAgent class, on the other hand is the parent class of all the service agent classes. More precisely speaking, the ServiceAgent class is the parent class of the CSGAgent class which in turn is the parent class of all the service agent classes. The ServiceAgent class maintains the basic property of all the service classes which includes defining the server host and the server port, receiving the various tasks delegated to the service agents, connecting to the task agents and execution of the jobs. This class also provides the GUI of the service agents. Three service agents are created for this tool viz. UNAgent which does the task of union, INAgent which does the task of intersection and DIAgent which does the task of difference. All these three service agents are extended from CSGAgent. The specific panel for creating geometric shapes is also included in each of the service agents which are finally added to the common gui in the ServiceAgent class.

- **IMPTask:** IMPTask is the abstract parent class for all activities which are carried out by the service agents. It contains the methods to update contents for each task and provides the string code necessary for each service agent to understand a task. This IMPTask class can be extended to define any specific task which is carried out by service agents. Further we have regrouped the tasks into CSGTask which is used for modifying the geometric shapes and FileReadTask which is used for reading different types of files. For this tool we have only implemented the CSGTask. The CSGTask class is extended to create the INTask which is used to perform the Intersection service, the UNTask which is used to perform the Union task, the DITask which is used to perform the Difference task. These tasks are defined by mathematical functions which are again derived from the DrawProject Java project. Before the operation is performed on each task, the geometric shape which is created on the service agents is moved over the shape which is sent from the GraphAgent. The logic behind moving each shape in the service agent's panel is again defined inside each of the service agent's class. The relevant class diagrams are given in Figure 4.

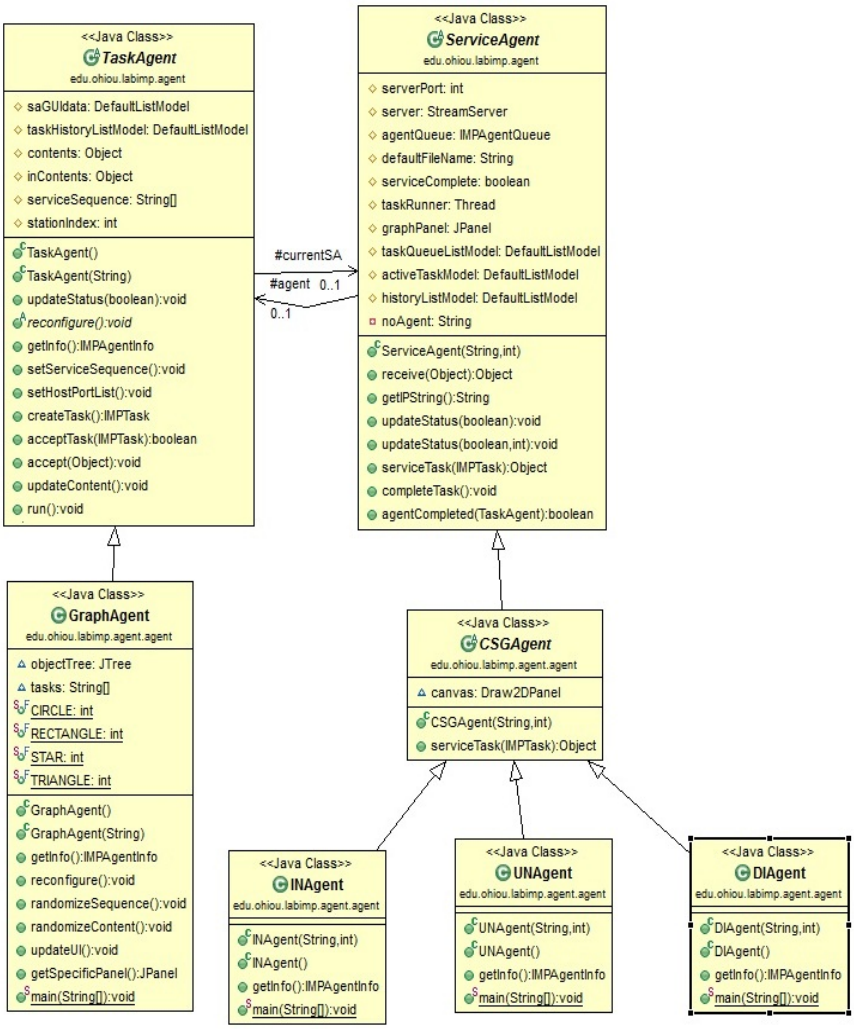

Figure 4- Relevant Class Diagrams

4.2 Agent's Initialization

Agent initialization start by running separate Java processes for Task Agent and Service Agents. There can be arbitrary number of each. We have to run the corresponding classes for each agent viz. GraphAgent class for initializing the task agent, the UNAgent class for the Union Service Agent, the INAgent class for the Intersection Service Agent, the DIAgent class for the Difference Service Agent. For starting the service agent we have to give input of the port number on which the service agent will listen for service requests and then click the start button to run the services. On the task agent's platform, we have to add the IP address of the computers along with their port numbers on which service agents may be listening. The list can include active ports with agents, but also can include any other ports and IP adresses. The second initialization step is to create the initial geometric shape which we like to send and then click the send button to run the Task agent and start the process. Figure 5a , 5b illustrates the process.

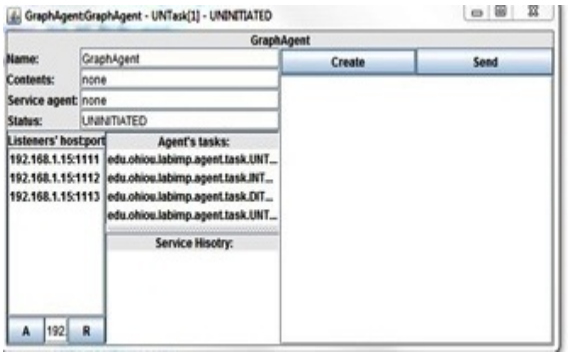

Figure 5a-Graph Agent

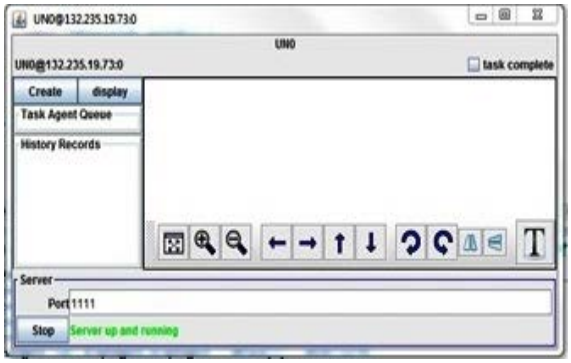

Figure 5b- Union Agent

4.3 Agent's Communication

The connection between Graph Agent and Service Agents are defined using the StreamClient class and the StreamServer classes which implement the Connectable interface. The Service agents and the Graph agents are located in different remote location each having an individual IP addresses. The Service before starting to be active takes the port number from the user and uses that port for receiving content from the Task agent. When the task agent searches for available service agents for performing the first service in its task list, for each IP/Port case, if connection is established, Task Agent requires for capabilities of the service agent, if the service matching its task is found, transfer is initialized. If first service does not exist for any IP/Port in the list, communication is attempted again after a timeout. When transfer is initialized Task Agent serializes its state and sends it to service agent memory space. This procedure is repeated automatically after completion of each task in task sequence until all tasks are completed, at which time task agent goes to COMPLETE state. If more than one service is available for a definite task, then the Task agent chooses the service agent whose host number is before the host number of the other available Service Agents.

4.4 Service Processing

The processing of services is explained with the help of the illustrative example in Figure 6. The GraphAgent, shown in the figure sets up request for three services, Union service from the UNAgent, Intersection Service from the INAgent and the Difference Service from the DI Agent. It will set up these tasks in sequenced order and add up the host IP address along with the port number for these particular services. It will also create the geometric shape on which these services are required and send it. So first it will search for UN Service, and once found it will go into the queue of the UNAgent. The UNAgent can accept multiple requests from different TaskAgents and put all the tasks in a queue and process them according to first come first serve basis. The UNAgent will also have its own geometric shape on which the shape coming from the TaskAgent will interact to carry out the operation. Once the service is done, the modified content will be sent to the next ServiceAgent which is in the queue. This process will continue until all the services are performed. In Figure 6, the TaskAgent has a job list of 5 tasks among which the second one and the fourth one is of same type and is carried on in the same Service agent. According to the job sequencing, the TaskAgent first goes to the first ServiceAgent , then to third ServiceAgent , to second ServiceAgent back to the third service agent and finally to the fourth service agent. A fifth service agent which is active remains in the idle state because of no requests.

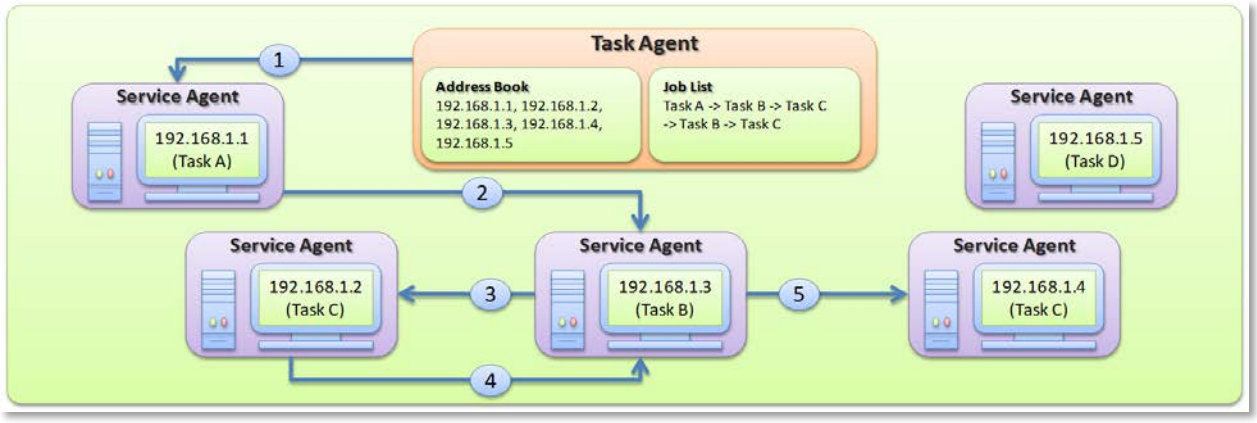

Figure 6- Job Sequencing Illustration

In our system, service processing perform an appropriate Boolean operation on the current shape of the GraphAgent using its own tool shape. In addition the tool has a moving shape in order to illustrate dynamic net use of the service processing. Sequence of tasks generates a tree of Boolean operations which is the result of each service. An example is shown in Figure 7.

5 Demonstrations and Result

For the purpose of demonstration of the education tool we have run different two Graph agents from different hosts and multiple service agents from other remote host locations. The two graph agents are ran from Host A and Host B. Host A also is running UNAgent, INAgent. Host B will be running INAgent and DIAgent. There is a third location Host C which will be running UNAgent , INAgent and DIAgent. Ofcourse, all these agents will be running from different ports in these hosts. For the GraphAgent running from Host A, we have added the IP address and port number of the UNAgents of Host A and Host C, the address of the INAgent of Host B and the address of the DIAgent of Host C. For the GraphAgent running from Host B, we have added the IP address and port number of the UNAgents of Host A, the address of the INAgent of Host B and Host C and the address of the DIAgent of Host C. Both the Task Agents have the tasks UN, IN and DI sequenced in order. The screenshots shown below illustrate the working of these service and task agents on different computers.

Firstly, the GraphAgent from Host A will search for a UN Service. Since it has two Host Services added up in the list, it will take the one with IP Address before the other one. So here it will take the Host A's UNService. The GraphAgent running from Host B, started will also get the UN service from host A. However, as GraphAgent from Host A has arrived first, it will get the service before the GraphAgent from Host B. Next GraphAgent from Host A will search for INAgent and find it in Host B, perform the task and then go for DIAgent in Host C, and finally complete its action. The GraphAgent from host B will, after completing the Un task will go to the INAgent in host B. If by that time Host A GraphAgent is done then it will start service immediately, otherwise, it will wait in a queue. Finally after completing the INTask it will go for DI task to DIAgent in Host C. If the DIAgent is still busy, it will wait or go directly and finish the job. The following figures show some of the operations.

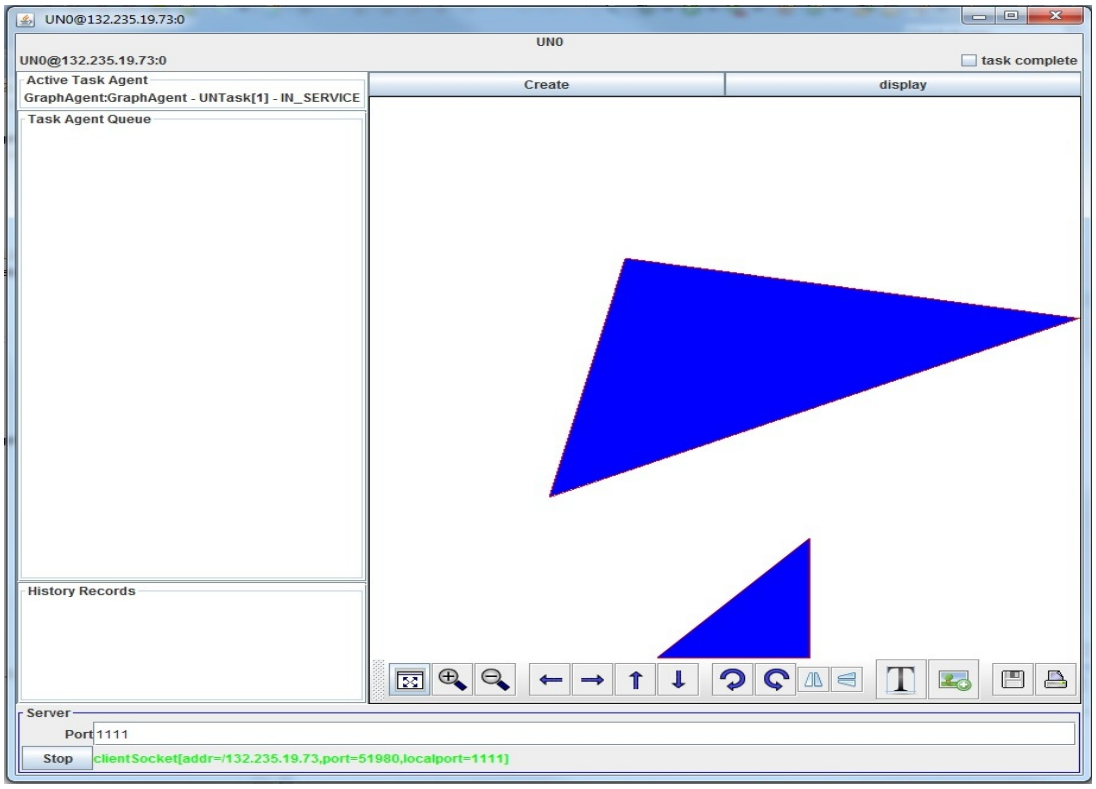

Figure 7a- Union Operator

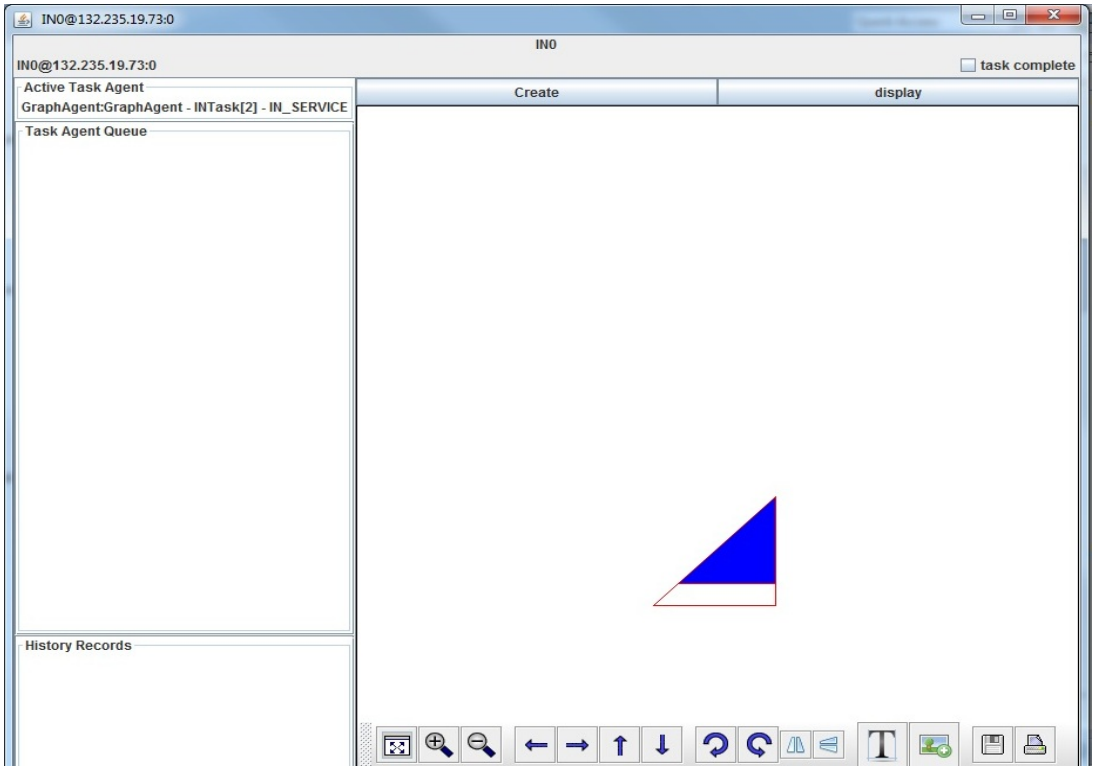

Figure 7b- Intersection Operator

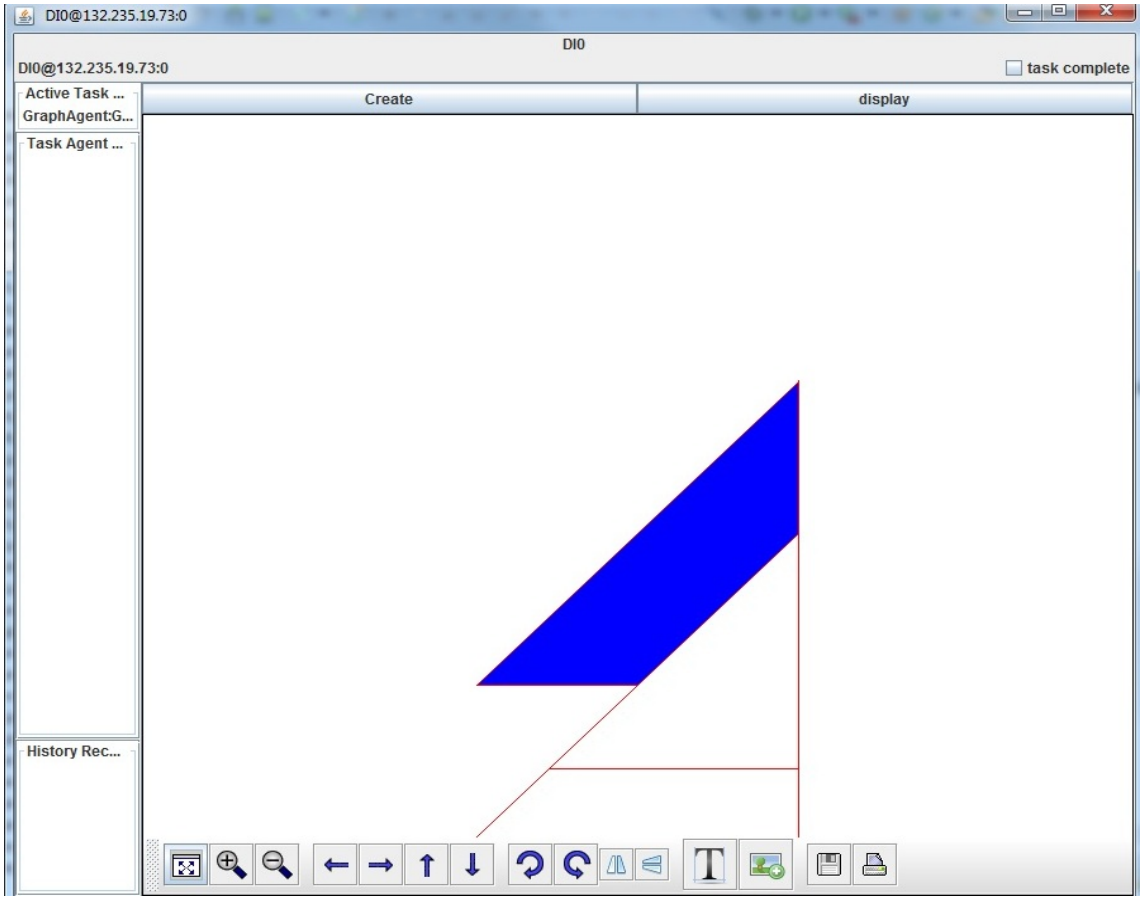

Figure 7c- Difference Operator

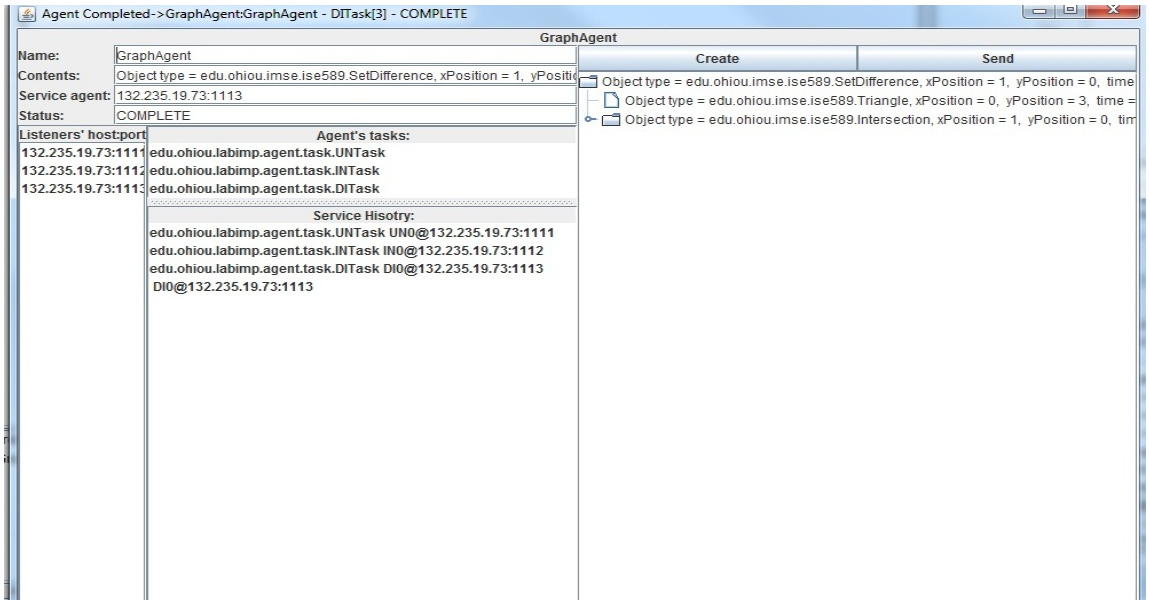

Figure 7d- Object Tree

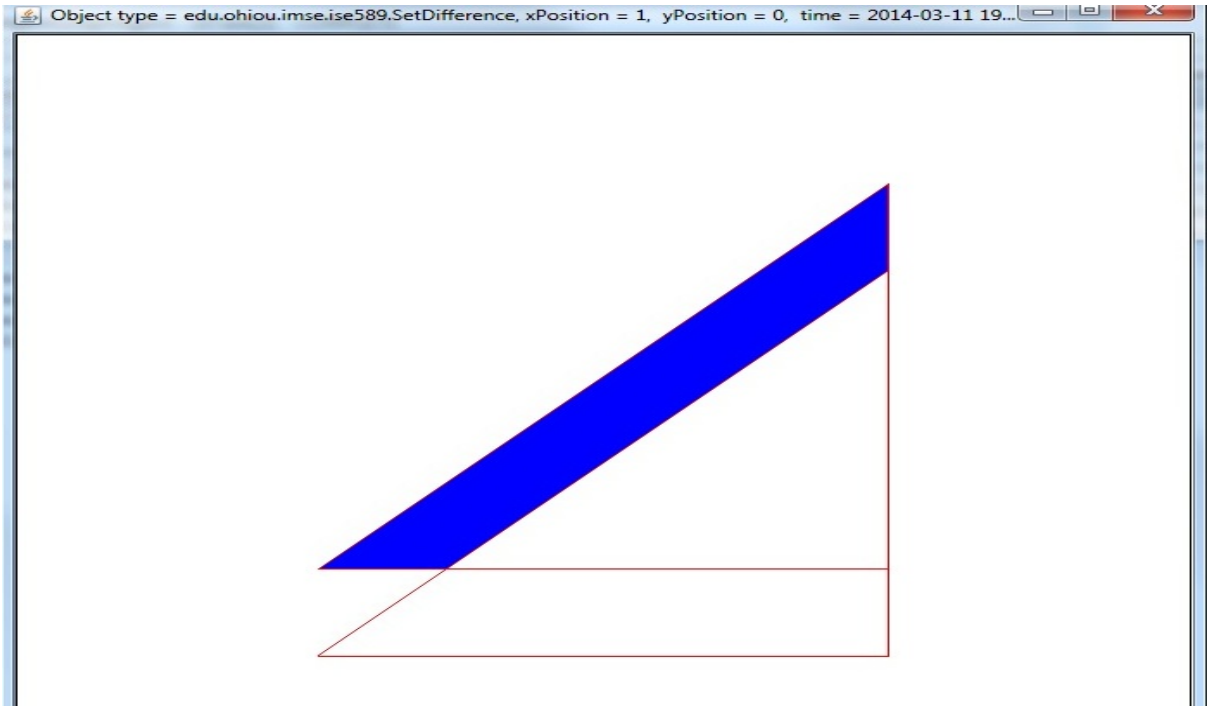

Figure 7e- Final Object

6 Conclusions

From the demo run, it is well evident that this is a very dynamic and autonomous network of distributed processing. We can further extend this network to include other types of services which can be used to read files or for feature recognition of mechanical parts. These services can also be used for running process planning, scheduling etc. Another issue which is observed while running different tasks from remote systems is that if the port number on which the Service is running is not same as the port number through which it is listening. The port number through which it is listening needs to remain open and not utilized by other services. For that we have introduced user input for selecting the listener's port number. However if the listening port number overlaps with the Service port number, it can cause a deadlock. We should keep this in mind while running the tool.

7 References

[1] The Foundation of Intelligent Physical Agents (FIPA), http://www.fipa.org/, accessed May 2006.

[2] Lange, Danny B. "Mobile objects and mobile agents: the future of distributed computing?." *ECOOP'98—Object-Oriented Programming*. Springer Berlin Heidelberg, 1998. 1-12.

[3] M. Fletcher, R.W. Brennan, D.H. Norrie, Modeling and reconfiguring intelligent holonic manufacturing systems with internet-based mobile agents, Journal of Intelligent Manufacturing, 14(1), pp. 7-23, 2003.

[4] Shen, W., Norrie, D.H., 1999, Agent-Based Systems for Intelligent Manufacturing: A State-of-the-Art Survey, *Knowledge and Information Systems, an International Journal*, 1(2), 129-156

[5] Wang L. H., Shen W. M., 2003, DPP: An agent-based approach for distributed process planning, *Journal Of Intelligent Manufacturing*, 14 (5): 429-439

[6] Dušan N Šormaz, Distributed Agent-based Integrative Model for Mass Customization Product Development, XV International Scientific Conference on Industrial Systems (IS'11), 2011

[7] Dusan N. Sormaz, Agent-based process sequencing using search algorithms, ASME International Conference on Manufacturing Science and Engineering, 2006

[8] Arkopaul Sarkar, Dusan N. Sormaz, Jing Huang, Multi Agent framework for design and manufacturing integration, *International Conference on Production Research- America 2012*